



UNIVERSIDAD AUTÓNOMA METROPOLITANA

MINERÍA DE DATOS CON SPARK y SCALA

Tesis que presenta
Sofía Ortiz Valenzuela
Para obtener el grado de
Maestro en Ciencias y Tecnologías de la Información

Asesor:

Dr. René Mac Kinney Romero

Jurado calificador:

Presidente: **Dr. Mario Graff Guerrero**

Secretario: **Dr. Pedro Lara Velázquez**

Vocal: **René Mac Kinney Romero**

México, D.F. Marzo 2017

Agradecimientos

Un agradecimiento a mi asesor el Dr. Rene Mac Kinney Romero por el apoyo, consejos y paciencia durante la realización de este trabajo.

A los asesores el Dr. Mario Graff Guerrero y Dr. Pedro Lara Velázquez por los comentarios aportados para el mejoramiento de esta tesis.

A la Universidad Autónoma Metropolitana - Unidad Iztapalapa y los académicos que han sido parte importante durante mi formación en la maestría.

Al Consejo Nacional de Ciencia y Tecnología, CONACyT, por la beca otorgada para estudios de posgrado.

Finalmente agradezco a mis padres y hermana por el apoyo mostrado a lo largo de esta maestría.

Contenido

Lista de Figuras	v
Lista de Tablas	viii
1. Introducción	3
2. Minería de datos	11
2.1. Descubrimiento de conocimiento de bases de datos (<i>KDD</i>)	12
2.2. Minería de datos	13
2.2.1. Función del modelo	14
2.2.2. Representación del modelo	14
3. Aprendizaje Maquinal	17
3.1. Inferencia de conceptos	18
3.1.1. Tarea de inducción: Sábados por la mañana adecuados para jugar tenis	20
3.2. Árboles de decisión	20
3.2.1. Algoritmo ID3	22
3.2.2. Bosques Aleatorios	27
3.3. Bayes Ingenuo	28
3.3.1. Teorema de Bayes	29
3.3.2. Descripción de Bayes Ingenuo	29
3.4. Vecinos más cercanos	33
3.5. k-Medias	38
3.5.1. Algoritmo <i>K</i> -medias	38
3.6. Apriori	39
3.6.1. Descubrimiento de conjuntos frecuentes	40
3.6.2. Algoritmo Apriori	41
3.7. Resumen	43
4. Big Data	45
4.1. Plataformas de almacenamiento de Big Data	46
4.2. Plataformas de programación	47
4.2.1. MapReduce	47

4.2.2. Spark	53
4.3. Resumen	58
5. Implementación de algoritmos en Spark	61
5.1. Bayes Ingenuo en Spark	62
5.2. k NN Join en Spark	65
5.3. Bosques Aleatorios en Spark	69
5.4. Apriori en Spark	70
5.5. k -Medias en Spark	73
5.6. Resumen	77
6. Bases de datos	79
6.1. Enron	79
6.2. <i>Record Linkage Comparison Patterns</i>	81
6.3. Hepmass	82
6.4. Reuters RCV1	83
7. Experimentos	87
7.1. Validación Cruzada	87
7.2. Métricas de evaluación	87
7.3. Equipo	89
7.4. Formato de pruebas	90
7.5. Resultados	91
7.5.1. ENRON	91
7.5.2. <i>RLCP</i>	95
7.5.3. Hepmass	97
7.5.4. Reuters_4CAT	99
7.6. Relación de resultados	102
8. Conclusiones	105
Referencias	109

Lista de Figuras

2.1.	Ilustración del proceso de descubrimiento de conocimiento (KDD)	13
3.1.	Ilustración de la partición por axis de un espacio de dos dimensiones.	22
3.2.	Árbol de decisión para la clasificación <i>JugarTenis</i>	22
3.3.	Vecinos más cercanos. Ejemplo de casos positivos y negativos con un punto x_q para clasificar.	36
3.4.	Todos los posibles conjuntos de los elementos: $\{0,1,2,3\}$	41
3.5.	Todos los posibles conjuntos de elementos, los de color gris son los conjuntos no frecuentes.	42
4.1.	Ilustración del modelo MapReduce. La función de mapeo es aplicada al conjunto original, lo que produce resultados intermedios. Que serán agrupados para ser procesados por las funciones de reducción	49
4.2.	Representación esquemática de MapReduce. En el ejemplo se muestra un <i>cluster</i> con tres computadoras con dos procesadores cada una.	50
4.3.	Arquitectura del Manejador de Recursos YARN. Que se encarga de ejecutar trabajos MapReduce.	52
4.4.	Ejemplo de como Spark calcula las etapas. Los rectángulos con líneas solidas son RDD's. Las particiones son los cuadros grises y negros. Se muestran 3 etapas generadas.	55
4.5.	Ejecución de Spark en un <i>cluster</i> . El programa principal asigna tareas a los nodos y cada nodo devuelve sus al programa principal.	57
5.1.	Representación del enfoque de calculo de histogramas en Spark. El conjunto de entrenamiento se divide en particiones, por cada una se obtiene su histograma local y en base a estos el global.	63
5.2.	Ilustración de la construcción de un árbol <i>Kd</i> de dos dimensiones.	68
6.1.	Ejemplo de un documento del corpus Reuters RCV1	85
7.1.	Validación cruzada con k bloques. Donde P=partición de pruebas, E=partición de entrenamiento.	88
7.2.	Resultados para la medida de desempeño de exactitud de ENRON.	92
7.3.	Resultados para los tiempos registrados para ENRON.	92

7.4. Resultados para los tiempos registrados para Apriori con 5,172 documentos.	94
7.5. Resultados para los tiempos registrados para Apriori con 33,716 documentos.	94
7.6. Resultados para la medida de desempeño de exactitud de <i>RLCP</i>	95
7.7. Resultados para los tiempos registrados para <i>RLCP</i>	96
7.8. Resultados para la medida de desempeño de exactitud de Hepmass	98
7.9. Resultados para los tiempos registrados para Hepmass	98
7.10. Resultados para la medida de desempeño de exactitud de Reuters_4CAT . . .	99
7.11. Resultados para los tiempos registrados para Reuters_4CAT	100
7.12. Resultados para los tiempos registrados para Apriori con 273,197 documentos.	101
7.13. Resultados para los tiempos registrados para Apriori con 628,333 documentos.	102

Lista de Tablas

1.1. Tamaño de unidades	6
2.1. Tareas dentro de la minería de datos	14
3.1. Ejemplos positivos y negativos del concepto <i>mamifero</i>	18
3.2. Ejemplos de hipótesis para el concepto <i>mamifero</i>	19
3.3. Día en el que se desea predecir si es adecuado para jugar tenis.	20
3.4. Ejemplos de entrenamiento para la clasificación del atributo <i>JuegaTenis</i> . .	21
3.5. Ejemplo de una lista de transacciones de compras hechas en una tienda de comestibles	39
4.1. Transformaciones sobre los RDD's	54
4.2. Acciones sobre los RDD's	54
6.1. Descripción Enron	80
6.2. Distribución <i>Record Linkage Comparison Patterns (RLCP)</i>	82
6.3. Cantidades de registros de Hepmass	83
6.4. Distribución de registros de Reuters_4CAT.	86
7.1. Matriz de confusión para clasificación binaria	88
7.2. Métodos para ejecutar	90
7.3. Medidas de calidad de los métodos sobre ENRON	92
7.4. Detalle de medidas de la clase = 'Spam'	93
7.5. Detalle resultados de conjuntos frecuentes de ENRON. D^* =Número de docu- mentos. N^* =Número de elementos diferentes. L^* =Cantidad máxima de ele- mentos por conjunto	93
7.6. Medidas de calidad de los métodos sobre RLCP	95
7.7. Detalle de medidas de la clase 'No Coincide'	96
7.8. Medidas de calidad de los métodos evaluados sobre Hepmass	97
7.9. Detalle de medidas de la clase 'Señal'	97
7.10. Medidas de calidad de los métodos evaluados sobre Reuters	99
7.11. Detalle de medidas de la clase 'MCAT'	100

7.12. Detalle resultados de conjuntos frecuentes de Reuters_4CAT. D^* =Número de documentos. N^* =Número de elementos diferentes. L^* =Cantidad máxima de elementos por conjunto	101
7.13. Relación de mejores resultados por base de datos	102

Capítulo 1

Introducción

El campo de la *minería de datos* ha presentado diversos avances en los últimos años. Se define a la minería como el proceso de encontrar *patrones* en los datos de forma automática. Lo cual tiene una larga y exitosa historia. Por ejemplo, Kepler a partir de observaciones registradas de los movimientos de los planetas logró inferir las leyes que llevan su nombre, las cuáles permiten describir y predecir el movimiento de los planetas. En la actualidad dichas observaciones consisten en registros de bases de datos. Donde es posible descubrir patrones, como puede ser: en un banco a partir de registros de clientes encontrar el patrón de aquellos aptos para un crédito y utilizarlo con nuevos clientes para determinar si otorgar o rechazar una solicitud.

Conforme las bases de datos crecen también incrementa el interés por encontrar *conocimiento* oculto en ellas. Este conocimiento se representa en forma de patrones, donde estos últimos se caracterizan por ser: novedosos, útiles (en el sentido que aporten algún beneficio) y ciertos (acorde a un criterio definido por el usuario). El enfoque tradicional para descubrir los patrones es llevar a cabo un análisis manual, sin embargo la tarea se complica cuando se tienen bases de datos de cientos de miles de registros. Para lo que se requieren técnicas y métodos que permitan extraer el conocimiento mediante el uso de computadoras. Siendo esto último el área de interés para el campo de minería de datos.

Como actividad la minería de datos consiste en explorar bases de datos en búsqueda de patrones. Tarea que no resulta trivial, ya que en la práctica la mayoría de las bases de datos reales no son diseñadas para recolectar datos para propósitos de análisis e interpretación. Por lo que es común que al mencionar minería de datos también se refiera al proceso de descubrimiento de conocimiento (*Knowledge Discoverey in DataBases KDD*), donde la minería de datos es un paso. Mientras que KDD es más amplio en el sentido que abarca una serie de tareas como: establecer objetivos, procesar los datos a un formato adecuados y validar resultados. La minería de datos es una tarea dentro de KDD que se enfoca en la selección y adecuación de técnicas para detectar los patrones de interés. Validar que tan interesantes o útiles son, se lleva a cabo en pasos posteriores; donde generalmente intervienen expertos en el dominio correspondiente.

En esencia los patrones que se busca encontrar mediante la minería de datos consisten en relaciones entre datos. Estas relaciones ayudan a explicar algo de los datos y también pueden ser utilizados para predecir el resultado de eventos ante nuevas situaciones. Es posible dividir las funciones de la minería en dos categorías: *descriptiva* y *predictiva*. Del lado descripti-

vo se encuentran técnicas que permiten descubrir patrones que muestren de forma explícita las relaciones entre datos, los cuales muestran introspectiva. Del lado de predicción, se encuentran los métodos enfocados a encontrar patrones que puedan ser utilizados para realizar predicciones. Las tareas más comunes dentro de la minería de datos junto con ejemplos de sus aplicaciones se listan a continuación:

- **Clasificación:** consiste en predecir la categoría a la que pertenece un elemento. Ejemplos incluyen: clasificación de correo electrónicos en válido o basura, sistemas expertos en la asistencia para la predicción de diagnósticos de enfermedades, etc.
- **Regresión:** permite predecir un valor numérico. Por ejemplo, predecir el valor de una casa considerando el valor de casas aledañas o con características similares.
- **Agrupación:** se enfoca en descubrir grupos de objetos a partir de características en común, cuando no se conocen los grupos de antemano. Ejemplo de aplicación: incluyen segmentación de clientes para aplicar estrategias de mercadeo dirigido.
- **Conjuntos frecuentes:** se utiliza para encontrar elementos que frecuentemente ocurren juntos. El análisis de la cesta en un ejemplo donde se aplica esta tarea, que consiste en encontrar los productos que frecuentemente se compran juntos.

Para encontrar los patrones de interés la minería de datos utiliza técnicas de una gran variedad de campos. Como algunos de los que se describen en la siguiente lista:

- **Estadística:** estudia recolectar, organizar e interpretar los datos. Las técnicas estadísticas se utilizan para encontrar las correlaciones y generar modelos. Estos últimos permiten describir objetos en términos de variables aleatorias y sus probabilidades de distribución asociada. Por ejemplo, dentro de minería de datos se utilizan estas técnicas para clasificación.
 - **Bases de datos:** se enfoca en la creación, mantenimiento y uso de base de datos. De forma general, este campo se encarga de la generación de modelos de datos, lenguajes de consulta, captura y almacenamiento de datos. En minería de datos facilita el acceso y manejo de los datos para realizar tareas de análisis más complejas. Recientemente ha surgido los almacenes de datos (*Data Warehouse*) cuya función es integrar datos de diferentes en formatos adecuados para propósitos de análisis, generalmente en empresas, para la toma de decisiones.
 - **Visualización:** son las técnicas utilizadas para crear tablas, imágenes, diagramas y otras formas para mostrar de forma intuitiva por personas patrones.
 - **Aprendizaje maquina:** es una rama de la inteligencia artificial que se encarga de desarrollar algoritmos que permitan a las computadoras aprender con base a experiencia. Estos algoritmos toman como insumo un conjunto de datos como fuente de experiencia y construyen conocimiento que representa el aprendizaje.
-

De la lista anterior uno de los grandes pilares es el aprendizaje maquina, el cual es utilizado como herramienta para encontrar patrones que no son triviales de detectar. En este trabajo nos enfocaremos en los métodos que ofrece este campo aplicando: clasificación, agrupación y conjuntos frecuentes con los algoritmos: Bayes Ingenuo, k NN Join, Bosques Aleatorios, k -Medias y Apriori; los cuales serán descritos en capítulos posteriores. Las aplicaciones del aprendizaje maquina son diversos, sin embargo son dos los principales. El primero involucra mejorar el funcionamiento de un programa con base a experiencia, como puede ser aprender a jugar damas chinas, donde se espera que un programa mejore su desempeño, es decir, gane un mayor número de partidas con base en la experiencia adquirida. Y el segundo, se utiliza para la adquisición de conocimiento en forma de conceptos. Por ejemplo, un sistema capaz de aprender las preferencias de noticias de un usuario y utilizar dicho conocimiento para sugerir nuevas noticias, siendo un ejemplo representativo de una tarea de clasificación. Este segundo enfoque es utilizado por la Minería de datos para detectar patrones en los datos.

El aprendizaje maquina se puede dividir en dos tipos: *supervisado* y *no supervisado*. En el primer tipo se tienen datos con un valor asociado (categoría) que se utilizan para aprender a predecir una categoría o un valor numérico. Ejemplos de este tipo de aprendizaje son: clasificación y regresión. En el segundo tipo, el no supervisado, no se conocen las categorías y tiene como propósito descubrir grupos con datos similares, un ejemplo de este tipo es la agrupación. Adicionalmente las tareas de aprendizaje se conforman por dos etapas: Entrenamiento y Pruebas. En la primera etapa se lleva cabo el aprendizaje donde se construye el conocimiento en forma de modelos y en la segunda etapa de pruebas se evalúan dichos modelos para determinar su validez. Un enfoque común es separar la fuente de experiencia, denominada conjunto de entrenamiento en dos disjuntos, el primero se utiliza para construir el modelo y el segundo para evaluación. Esto con el propósito de evaluar que tan preciso resulta dicho modelo ante nuevas situaciones.

Considerando los dos tipos de aprendizaje estos son adoptados por la minería de datos para dos de sus funciones principales: la descripción y la predicción. Mientras que la minería de datos para propósitos de predicción selecciona algoritmos de aprendizaje supervisado. Del lado descriptivo selecciona algoritmos del aprendizaje no supervisado que describen las estructuras o relaciones entre los datos.

En la actualidad las tareas de minería se realizan sobre bases de gran tamaño. Se estima se generen 2.5 quintillones de bytes diariamente en el mundo. Para dimensionar esta cantidad en unidades de almacenamiento más entendibles en la tabla 1.1 se muestra una relación de equivalencias. Donde 1 quintillón= $10^{30}=1,000,000,000,000,000,000,000,000,000,000$.

Una de las razones más evidentes de la generación de estas cantidades colosales de datos es el desarrollo de la tecnología. Conforme los precios bajen y las capacidades de los dispositivos aumenten resulta más fácil el acceso a equipos que permitan digitalizar datos. Ejemplo de estos dispositivos incluyen: computadoras personales, teléfonos inteligentes, cámaras electrónicas, sensores integrados, etc. Dispositivos que en la actualidad se usan en casi todos los aspectos de nuestras vidas y que producen datos de una gran diversidad de acciones como: generación de correos electrónicos, compras en línea, transacciones bancarias, navegación web, ubicación geográfica, etc.

Medida	Notación	Tamaño en bytes
Gigabyte	10^9	1,073,741,824
Terabyte	10^{12}	1,099,511,627,776
Petabyte	10^{15}	1,125,899,906,842,624
Exabyte	10^{18}	1,152,921,504,606,846,976
Zettabyte	10^{21}	1,180,591,620,717,411,303,424
Yottabyte	10^{24}	1,208,925,819,614,629,174,706,176

Cuadro 1.1: Tamaño de unidades

Estas grandes cantidades de datos a dado origen al término *Big Data*, que comúnmente se utiliza para referirse a grandes colecciones de datos. Para explicar el concepto de Big Data generalmente se utiliza el modelo de las 3V's [22]: *Volumen*, *Variedad* y *Velocidad*. El término volumen se refiere a colecciones de gran tamaño, que hoy en día se expresan en términos de gigabytes hasta petabytes, dependiendo del contexto de la aplicación. Velocidad indica el tiempo en los datos son generados y los tiempos que toma realizar su procesamiento correspondiente. Variedad describe los diferentes tipos de datos (estructurados y no estructurados) así como las diversas fuentes. Posteriormente al modelo se incluyó la cuarta V de *Valor* que se usa para indicar la información valiosa que se puede extraer[16]. Y de forma más específica se puede describir a Big Data como bases de gran tamaño de diferentes tipos de datos que cuyo procesamiento sobre pasa las capacidades de las herramientas tradicionales [6, 18, 19](p.j. manejadores de bases de datos, sistemas de archivos, etc).

Es cada vez más común que Big Data se presente en diversos sectores. En el sector comercial un ejemplo es Walmart que se reporta almacena 4 petabytes de información. Las compañías de internet constantemente se enfrentan a problemas de Big Data, como es Facebook que almacena 600 terabytes diarios. En el sector de investigación, el centro de simulaciones climáticas de la NASA (*NCSS*, por sus siglas en inglés) mantiene 32 petabytes de datos de observaciones climáticas y simulaciones. Estos son algunos ejemplos de colecciones de datos de grandes compañías, sin embargo Big Data no sólo atañe a estas, ya que también es posible encontrar bases de datos dentro de su categoría en repositorios públicos como: UCI[43], Yahoo Research[36], Amazon AWS[3], etc; que pueden ser utilizados como puntos de referencia para propósitos de experimentación. Lo que muestra la clara presencia de las grandes colecciones de datos.

Big Data representa varios retos y oportunidades. Por el lado de los retos se encuentran problemas como: almacenamiento, análisis, visualización, etc. Los problemas se presentan debido a que los mecanismos y técnicas utilizadas fueron diseñados bajo la existencia de una menor cantidad de datos, por lo que hay una gran cantidad de esfuerzos enfocados a idear o adaptar nuevos métodos para operar bajo los requerimientos de Big Data. Del lado de las oportunidades en el sector empresarial ofrece beneficios como incrementos en la productividad, mejorar servicios a clientes, identificar nuevos productos y servicios. En el campo de la investigación el análisis de simulaciones de fenómenos naturales pueden llevar a entender el origen de las cosas.

Una de las tendencias para el procesamiento de Big Data es el computo distribuido. Han surgido plataformas que permiten acceder a los recursos del cluster para este propósito. Un cluster consiste en computadoras (nodos) conectadas vía red que se comportan como si se tratase de una. Estas plataformas permiten realizar cálculos en paralelo mediante una serie de operativas de alto nivel, sin que el usuario tenga que preocuparse por implementar los mecanismos referentes a la asignación de tareas, tolerancia a fallas, etc. Una de las herramientas pioneras y más adoptadas es Hadoop, la cuál es una implementación del modelo computacional *MapReduce*.

MapReduce es un modelo y marco de trabajo creado por Google para procesar enormes colecciones de datos. Se basa en el método de *divide y vencerás*, en donde un conjunto de datos que no podría ser manejable por una sola computadora, se divide en bloques más pequeños los cuales si pueden ser operados por una. La plataforma se encarga de distribuir de forma automática dichos bloques sobre los nodos del *cluster* para ser procesados de forma paralela, posteriormente los resultados parciales se combinan para obtener el resultado final. Para implementar el método de divide y vencerás el modelo ofrece dos funciones: *mapeo* y *reducción*.

En MapReduce el procesamiento de los datos se realiza por etapas. En la primera etapa se ejecutan las funciones de tipo mapeo seguida de una segunda etapa donde, se invocan las funciones de reducción. No hay comunicación entre tareas y el único momento donde se comparten datos es entre etapas. La política de MapReduce es almacenar en disco duro los resultados intermedios para futuros accesos. En aplicaciones donde se requiere de múltiples etapas dicha política podría impactar en los tiempos de procesamiento. Motivando el desarrollo de otras plataformas del mismo tipo pero que operan sobre memoria, característica que resulta conveniente para tareas de análisis que requieren realizar consultas interactivas, una de estas es *Spark*.

Spark es otra plataforma de computo distribuido, que para el procesamiento de los datos ofrece una abstracción de memoria distribuida llamada Conjuntos Distribuidos Resilientes (*Resilient Distributed Datasets, RDD's*) y operaciones sobre estas. Los RDD's son una colección de solo lectura distribuida. Se pueden comparar con una colección de datos como la que se encuentran en los lenguajes de programación con el conocimiento previo de que se trata de una colección distribuida y puede ser operada en paralelo. La forma de operar es la siguiente, inicialmente el conjunto de datos de entrada se divide en particiones y cada uno de estas es asignada a un nodo dentro del *cluster*. Para esto último los datos se leen y se representan mediante un RDD, en donde cada nodo ofrece espacio de memoria para albergar una parte de la colección. Posteriormente es posible aplicar operaciones sobre dicho RDD y los resultados se depositan en uno nuevo. Es posible observar que los resultados se mantienen en memoria, mediante los RDD's, disponibles para futuras consultas; agilizando su acceso por el costo adicional que puede requerir mantener la colección.

Este tipo de plataformas se caracterizan por encargarse de las tareas pertinentes al paralelismo así como distribución de tareas, y Spark no es la excepción. Donde esta última de forma automática se encarga de: calendarización de tareas, tolerancia a fallas y escalabilidad. Además, cuenta con una implementación de código abierto con el mismo nombre denominada

Apache Spark.

En este trabajo nos enfocaremos en realizar tareas de minería de datos sobre grandes cantidades utilizando métodos de aprendizaje maquina. Como se mencionó anteriormente, las técnicas de análisis así como los métodos de aprendizaje maquina fueron desarrollados bajo la premisa de la existencia de pocos datos, condición que no cumplen las colecciones de Big Data. Por lo que se tiene por objetivos los siguientes:

- I. Describir la forma en la que se pueden llevar a cabo tareas de minería de grandes volúmenes de datos sobre la plataforma Spark.
- II. Así como conocer a Spark para determinar que tan efectiva es para estos propósitos.

Spark se enfoca al procesamiento intensivo de datos, para propósitos de minería se requiere expresar los algoritmos correspondientes en las operaciones que ofrece. Siendo requerido aprender la herramienta así como las operaciones que ofrece, y ser capaz de expresar los algoritmos en términos de estas. Para abordar el primer objetivo se seleccionaron 5 procedimientos de aprendizaje maquina correspondientes a tareas de minería más comunes mencionadas anteriormente, el listado de los procedimientos asociados a la tarea se muestra a continuación

- Clasificación: Bayes Ingenuo, Bosques Aleatorios, k NN.
- Agrupación: k -Medias.
- Conjuntos frecuentes: Apriori.

Adicionalmente Spark ofrece una librería que permite hacer aprendizaje maquina denominada *MLlib*. La cual contiene implementación de aproximadamente 20 algoritmos de aprendizaje. Entre los que podemos encontrar a: Bayes Ingenuo, Bosques Aleatorios y k -Medias. Donde los métodos que ofrece corresponden a las versiones en MapReduce. Siendo *MLlib* un proyecto activo que permiten hacer aprendizaje sobre Big Data. Por lo que se utilizó dicha librería para conocerla y realizar comparaciones con los métodos desarrollados.

Considerando que es posible expresar el modelo MapReduce en Spark y que las versiones en *MLlib* se basan en este. Para llevar a cabo las implementaciones de los métodos se tomó como punto de partida dicho modelo así como la versión tradicional. Fue durante el desarrollo que se realizaron las adaptaciones correspondientes sobre Spark y los RDD's.

Para el caso del segundo objetivo, se evaluarán los procedimientos sobre bases de datos que cumpla características de Big Data. En este caso fueron 4 bases de datos seleccionadas: Enron, RCLP, Hepmass y Reuters. Se mostrarán los resultados de la evaluación tanto de los métodos implementados así como los disponibles en *MLlib*, utilizando medidas de desempeño ampliamente utilizadas: Exactitud, Coeficiente de correlación de Matthews, Precisión, Memoria y Medida-F.

La organización de este trabajo es de la siguiente forma. En el capítulo 2 se explican conceptos de minería de datos, así como el proceso de descubrimiento KDD, el cual fue utilizado como metodología en la etapa de pruebas para la evaluación de los resultados. En

el capítulo 3 se revisarán los procedimientos de aprendizaje maquina seleccionados. En el capítulo 4 se abordará con detalle la forma de operar de las plataformas MapReduce y Spark. En el capítulo 4 se muestra el desarrollo de los algoritmos implementados en Spark. En el capítulo 5 se describen las bases de datos utilizadas, así como las etapas del proceso KDD aplicadas a cada una y los resultados de la evaluación de todos los métodos implementados tanto como las versiones de MLlib. En el capítulo 6 se muestran conclusiones así como trabajo a futuro posible a realizar.

Capítulo 2

Minería de datos

En la actualidad vivimos rodeados de datos. En los últimos años han surgido una gran cantidad de dispositivos digitales a un costo accesible, lo que ha facilitado el acceso a equipos que permiten digitalizar información en forma de datos. Ejemplos de estos equipos son: computadoras, teléfonos inteligentes, cámaras digitales, etc. El uso de dispositivos digitales en casi todos los aspectos de nuestra vida diaria permiten registrar en forma de datos una gran cantidad de nuestras acciones. Acciones que se registran diariamente incluyen: compras hechas en tiendas o por internet, publicaciones en redes sociales, transacciones bancarias ya sea en cajeros automáticos o en línea, ubicaciones geográficas por medio de teléfonos inteligentes, etc.

Oculto en estos datos existe conocimiento en forma de patrones que puede ser utilizado para obtener algún beneficio. Por ejemplo, a partir de una base de datos con registros históricos de compras de clientes es posible detectar los patrones de compra de diferentes grupos demográficos. Un caso muy citado, es el descubrimiento que los Sábados por la noche los hombres que compraban cerveza compraban pañales. Este es un ejemplo sencillo de un patrón de compra que puede ser utilizado para establecer estrategias de mercadeo dirigido con el propósito de aumentar las ganancias.

La búsqueda de patrones a partir de datos ha dado origen al campo de *minería de datos*. Las bases de datos actuales contienen miles o millones de registros, por lo que la tarea de analizarlas para detectar algún patrón interesante puede resultar una tarea abrumadora para una persona. La minería de datos utiliza métodos computacionales para realizar el descubrimiento automático de patrones.

La minería de datos se enfoca en descubrir patrones complejos e interesantes. Estos patrones se representan como modelos que encapsulan relaciones entre los datos. Se consideran complejos si no resulta tan trivial identificar estas relaciones. Para detectarlos la minería de datos utiliza algoritmos y técnicas de campos como: aprendizaje maquina, estadística, inteligencia artificial y bases de datos.

Es común que al hablar de minería de datos se hable de *descubrimiento de conocimiento en bases de datos* (KDD por sus siglas en inglés). En la práctica la mayoría de los datos no son registrados para propósitos de minería. Ante esta situación, el descubrimiento de conocimiento considera tareas adicionales como: acceso y transformación de los datos para el descubrimiento de patrones. Uno de estas tareas es la minería de datos que se enfoca en aplicar algoritmos a los datos para descubrir algún patrón. El conjunto de estas tareas

conforman el proceso KDD que se explica con detalle a continuación.

2.1. Descubrimiento de conocimiento de bases de datos (*KDD*)

Las bases de datos contienen información valiosa que representa conocimiento en forma de patrones. Por ejemplo, a partir de una base de datos con solicitudes de crédito que con tienen información de clientes (ej. nombre, edad, salario,) y estatus de la solicitud (ej. pagos en tiempos/ pagos fuera de tiempo) se podría obtener el patrón de los clientes que si pagan a tiempo. Dicho patrón podría ser aplicado con nuevos clientes para predecir con un grado de certeza si un cliente puede pagar en tiempo y ofrecerle un crédito. Un ejemplo de patrón de esta aplicación sería clientes: edad ≥ 25 y salario $\geq 10,000$ pagan a tiempo = si, con un grado de certeza de un 80 %.

Es posible encontrar una gran cantidad de patrones en una base de datos, sin embargo el descubrimiento de conocimiento busca encontrar aquellos que cumplan las siguientes características: *novedosos*, *ciertos* (en base a un criterio establecido), *útiles* y *no triviales*. Un patrón novedoso es aquel que provee información nueva dentro del contexto de un problema. Debe ser valido para nuevos datos con cierto grado de certeza. Debe ser útil en el sentido que aporte algún beneficio, puede ser económico o ventaja. Y no debe ser trivial detectarlos, por lo que se requiere de técnicas que permitan inferir de los datos y tengan cierto grado de autonomía para realizar cálculos. Los patrones que cumplen con las cuatro características se considera conocimiento.

Extraer patrones en forma de conocimiento de una base de datos no es una tarea trivial. El enfoque más sencillo es aplicar técnicas que permitan encontrar patrones a partir de datos. Sin embargo, el descubrimiento de conocimiento va mucho más allá, considerando que los resultados de estas técnicas se ven afectados por los datos. Por lo que se requiere realizar tareas adicionales para la selección y adecuación de los datos con el propósito de obtener mejores resultados. Adicionalmente se requiere evaluar estos últimos para determinar su grado de veracidad así como su representación final.

La extracción de conocimiento consiste en un proceso [10] que inicia a partir de una base de datos y devuelve un modelo que describa a los datos o pueda ser utilizado para realizar predicciones. Los primeros pasos se enfocan en la selección y transformación de datos, los siguientes en la extracción de los patrones y los últimos en la interpretación de estos. En la imagen 2.1 se muestra el proceso KDD. Este es un proceso iterativo cuyo grado de avance esta definido por el descubrimiento. Los pasos que lo componen se detallan a continuación.

En la imagen 2.1[10] se muestran de forma general el proceso KDD y sus pasos se describen a continuación:

1. **Limpieza:** son tareas básicas de procesamiento de datos como: eliminación de ruido y valores atípicos, manejo para la falta de valores y datos dinámicos (series de tiempo).
-

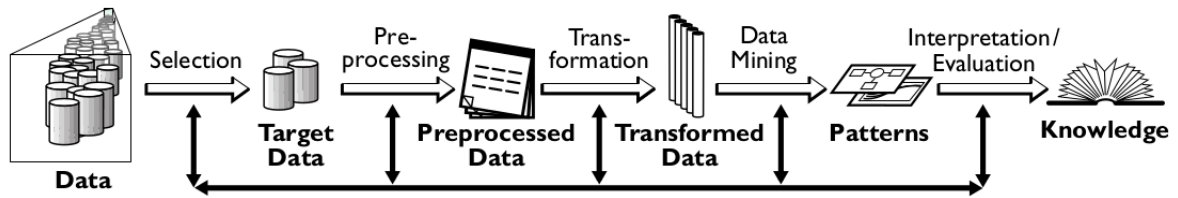


Figura 2.1: Ilustración del proceso de descubrimiento de conocimiento (KDD)

2. **Selección:** se enfoca en seleccionar características que aporten mayor información. Así como muestras de instancias.
3. **Procesamiento Previo:** donde múltiples fuentes son unificadas.
4. **Transformación:** procesar los datos de entrada en un formato adecuado para los algoritmos de minería.
5. **Minería de datos:** Es la selección y aplicación de algoritmos de minería de datos para extraer patrones (modelos). La elección estará en función de: la tarea de minería de datos (ej. clasificación, regresión, agrupación, etc.), la naturaleza de los datos (algunos algoritmos funcionan únicamente con datos categóricos otros con datos continuos y algunos ambos) y la aplicación (ej. el usuario puede estar interesado en obtener un modelo de fácil comprensión más que en sus capacidades de predicción).
6. **Interpretación y evaluación:** incluye interpretar los patrones descubiertos y con base a esto repetir alguno de los pasos anteriores. Así como descartar patrones redundantes o irrelevantes, así como traducir a términos entendibles por usuarios.

Cada uno de los pasos es importante para el éxito del descubrimiento. El modelo no será valido si los datos no son recolectados y procesados de manera adecuada o la formulación del problema no es correcta. En el presente trabajo se enfoca principalmente en el paso *minería de datos*, que se describe con mayor detalle a continuación.

2.2. Minería de datos

Como se mencionó anteriormente, la minería de datos es un paso dentro del proceso KDD cuyo objetivo es generar un modelo a partir de datos. En dicho paso se asume que los datos tienen un formato adecuado, por lo que se enfoca primordialmente en la selección y aplicación de algoritmos para construir modelos. En caso de requerirse, también se encarga de la adecuación de algoritmos para: manejo de grandes cantidades de datos, bases de datos complejas, etc. Utiliza métodos de diferentes campos para generar los modelos como: aprendizaje maquina, estadística, inteligencia computacional, etc. Al tener como base una gran variedad de disciplinas, la lista de algoritmos es larga, y la elección se basa principalmente

en la *función del modelo* y su *representación*. Adicionalmente se toman en cuenta otras consideraciones como: la complejidad, naturaleza de los datos, tiempo de generación del modelo, precisión, etc.

2.2.1. Función del modelo

Los tipos de modelos de forma general se pueden dividir acorde a su función en dos tipos: *predicción* y *descripción*. La predicción permite realizar inducción sobre los datos y producir un modelo que sea utilizado para realizar predicciones. Por ejemplo: a partir de registros clínicos construir un modelo que permita predecir, ante un nuevo caso, el diagnóstico de una enfermedad. Mientras que la descripción se enfoca en encontrar estructuras que describan los datos. Retomando el ejemplo del diagnóstico, si se desconocen las enfermedades de interés, mediante este tipo de análisis se podrían describir por sintomatología nuevos tipos de enfermedades. En ocasiones la función de descripción precede a la de predicción. Las tareas más comunes que se llevan a cabo acorde a estas funcionalidades se listan en la tabla 2.1

Predicción	<p>Clasificación es la tarea de predecir a que categoría pertenece el nuevo dato. Un ejemplos es un programa que clasifique correo electrónico como “legítimo” o “basura”.</p> <p>Regresión permite la predicción de un valor numérico.</p>
Descripción	<p>Agrupamiento descubre de grupos y estructuras de datos que compartan características las cuales no se conocen de antemano. Un ejemplo de aplicación es la segmentación de consumidores en grupos similares para mercadotecnia dirigida.</p> <p>Análisis de asociación busca relaciones entre variables. Una aplicación es en el análisis de compras en tiendas de alimentos, donde se puede determinar que productos se compran juntos frecuentemente y utilizar ésta información para mercadotecnia. Un ejemplo muy mencionado es el descubrimiento que los clientes que compraban pañales también compraban cerveza.</p>

Cuadro 2.1: Tareas dentro de la minería de datos

2.2.2. Representación del modelo

Los modelos generados deben presentarse en una forma adecuada para el usuario final. Representaciones adecuadas para personas incluyen: estructura lógicas (ej. reglas y árboles de decisión), descripciones visuales, etc. En ocasiones los modelos serán utilizados por otros

programas, como son los sistemas expertos o sistemas de recomendación, en tales casos la representación pueden ser formalismos declarativos.

La representación determina la flexibilidad del modelo para expresar las relaciones entre los datos. Existen dos extremos en la representación de un patrón como: una caja-negra (*black-box*) cuya forma es incomprensible, como los generados por las redes neuronales; y los transparentes que muestran la estructura del patrón, que puede ser interpretado y ayuda a explicar algo de los datos. Modelos más complejos tienden a ser más precisos pero son más difícil de entender. La mayoría de las aplicaciones tienden a elegir modelos más sencillos debido a que son robustos y fáciles de interpretar.

Una lista de los 10 algoritmos más populares de minería de datos se describe en [48]. De los cuales la gran mayoría corresponde a clasificación, el resto incluye métodos de agrupación, regresión, análisis de asociación y grafos. Los diferentes algoritmos de clasificación revisados en el artículo refleja la diversidad de enfoques para resolver dicho problema. Por ejemplo, realizar clasificación mediante árboles de decisión (los cuales se explicarán posteriormente) puede ser útiles por que dan estructura a los datos y no tienen problemas para manejar bases con muchas características, además que pueden operar con valores continuos y discretos. Sin embargo, pueden no dar buenos resultados en caso de que los datos no sean separables de forma lineal. Por lo que no existe una técnica que de buenos resultados de forma general[47].

La minería de datos utiliza técnicas de aprendizaje maquinal para el descubrimiento de patrones. Estas técnicas permiten inferir modelos complejos a partir de datos, lo que se conoce como aprendizaje. En el siguiente capítulo se describe el aprendizaje maquinal.

Capítulo 3

Aprendizaje Maquinal

Para resolver un problema en una computadora se necesita un programa con las instrucciones específicas de las acciones que debe realizar para devolver la solución. Por ejemplo, que dada una colección de números obtener cuál es el mayor. Es posible desarrollar un algoritmo de esta tarea debido a que se conoce un procedimiento para obtener dicho número. Los componentes para este problema son: una lista de números, un algoritmo y un número de salida que corresponde al mayor de los de la lista de entrada. Por otro lado existen aplicaciones en las que no se conoce de antemano los algoritmos. Por ejemplo, asignar la categoría “legítimo” o “basura” a un correo electrónico. No se podría programar explícitamente las instrucciones para realizar la tarea, ya que no tenemos el conocimiento necesario. Lo que si se tiene de antemano es una colección de correos ya etiquetados con las dos posibles categorías.

Existen métodos que permiten a las computadoras inferir el conocimiento requerido para realizar una acción, actividad que si se realizará por una persona se identificaría como aprendizaje. En este sentido y retomando el ejemplo anterior, la computadora tendría que aprender lo que es un correo “legítimo” y uno “basura” a partir de correos anteriores, y ser capaz de establecer de forma autónoma el conocimiento para realizar la tarea.

El campo del Aprendizaje Maquinal, se encarga de desarrollar algoritmos que aprendan con base a experiencia. Es un campo multidisciplinario que tiene sus bases en: inteligencia artificial, estadística, ciencias de la computación, etc. Sus esfuerzos iniciales se enfocaron en aplicaciones que mejoraran su funcionamiento en una tarea con base a experiencia, es decir se adaptaran a nuevas situaciones para las que no fueron programadas con el propósito de mejorar su función. Por otro lado, surgió la aplicación de descubrir conocimiento en forma de conceptos en base de datos. Enfoque utilizado por la minería de datos para descubrir regularidades implícitas.

En la actualidad el aprendizaje maquinal está presente en una gran cantidad de aplicaciones. Por ejemplo, programas que permiten reconocer personas y etiquetarlas de manera automática, servicios de correo que en base a nuestras preferencias y su contenido asignan estos a la bandeja pertinente, tiendas en línea que muestran sugerencias de productos tomando en cuenta gustos personales, sistemas expertos que apoyan a la tarea de diagnóstico, etc. Aplicaciones que tienen en común el aprendizaje de los datos para realizar su función. Este tipo de aprendizaje consiste en generalizar a partir de ejemplos imitando a las habilidades cognitivas de los humanos. Y aunque no se ha logrado imitar por completo, si se han desarrollado métodos que permiten inferir conceptos de manera automática.

3.1. Inferencia de conceptos

Llevar a cabo la inferencia a partir de datos es posible mediante las técnicas que ofrece el aprendizaje maquina. Las cuales con base en un concepto objetivo que se encuentra descrito por atributos, que capturan diferentes valores en registros. Para explicar la inferencia considere aprender el concepto “Mamífero”. En la tabla 3.1 se muestran los ejemplos, donde la columna “mamífero” es el concepto que se desea aprender, el valor “si” implica que el animal en cuestión se trata de un mamífero. El resto de las columnas describen al concepto y se denominan atributos descriptivos o características. Cada registro llamado también instancia, representa un animal. En la tabla 3.1 es posible observar 3 ejemplos positivos (Mamífero=Si), de los cuáles la “Sangre=caliente”. Por lo que de forma trivial se podría establecer la hipótesis: *si sangre=caliente entonces mamífero=Si*.

Nombre	Sangre	Pare	Vuela	Vive en agua	Mamífero
Humano	Caliente	Si	No	No	Si
Ballena	Caliente	Si	No	Si	Si
Pitón	Fría	No	No	No	No
Murciélagos	Caliente	Si	Si	No	Si

Cuadro 3.1: Ejemplos positivos y negativos del concepto *mamífero*

Para este ejemplo, la hipótesis se representa como un vector con la conjunción de los atributos descriptivos del concepto y los posibles valores que puedan tomar cada uno de ellos. Los valores que pueden tomar los atributos son:

- ‘?’ que implica que se puede tener cualquier valor.
- Especificar un valor, específico que pueda tomar. Por ejemplo: “no” para el atributo “Vuela”.
- \emptyset , para indicar que ningún valor es aceptable.

Si existe al menos una instancia x que satisfaga todos los criterios de una hipótesis h' , entonces se considera valida. La hipótesis indicada anteriormente se podría representar de la forma que se indica en la figura 3.2, la cual es valida y positiva para el concepto objetivo “mamífero”. Dado que cada hipótesis se representa como la combinación de atributos descriptivos y sus posibles valores se puede tener una gran cantidad. La tarea de aprendizaje consiste en buscar aquellas validas en el conjunto dado. En la práctica solo se tiene acceso a una porción del universo total de ejemplos. Por lo que se dice que la hipótesis seleccionada $h(x)$ es una aproximación a la función ideal $f(x)$. Función que se obtiene considerando el conjunto completo y permanece desconocida.

ID	Sangre	Pare	Vuela	Vive en agua	Mamífero
?	Caliente	?	?	?	si

Cuadro 3.2: Ejemplos de hipótesis para el concepto *mamífero*

Para representar los problemas de aprendizaje, se utilizará la siguiente nomenclatura. El conjunto de registros sobre los cuales está definido el concepto objetivo se describe por X , mientras que cada instancia se identifica como x . En el ejemplo, X es el conjunto de animales representado por los atributos: *sangre*, *pare*, *vuela* y *vive en el agua*. El concepto o función que se desea aprender, se denota con $h(x)$, en general es una función binaria definida sobre las instancias X cuyo dominio de salida corresponde a los valores del atributo objetivo $Y = \{0, 1\}$, representado como, $f : X \rightarrow Y$. En el ejemplo, el atributo “mamífero” corresponde al concepto objetivo y Y a los valores que este puede tomar, esto es, $f(x) = 1$ si “mamífero”=sí y $f(x) = 0$ si “mamífero”=no. Por último, al conjunto de instancias utilizado para construir la función objetivo se identifica como D y se conoce como el conjunto de entrenamiento. Es común que las tareas de aprendizaje las instancias del conjunto D se les de el formato de tupla $\langle x, c(x) \rangle$, para separar los atributos descriptivos del concepto a aprender.

En base a la nomenclatura es posible establecer los componentes de una tarea de aprendizaje como[31]:

- Instancias X : ejemplos positivos y negativos que describen el concepto mediante atributos.
- Hipótesis.
- Concepto objetivo: $X \rightarrow \{0,1\}$.
- Conjunto de entrenamiento D .

Encontrar:

- Función objetivo $f(x)$. Hipótesis $h \in H$ tal que $h(x) \equiv f(x)$ para todo x en X .

El resultado de un método de aprendizaje maquinal es la función $h(x)$ que representa un modelo, este evalúa una instancia x y produce la salida y . La forma precisa de $h(x)$ se determina durante la fase de *entrenamiento*, también conocida como la fase de *aprendizaje*, a partir del conjunto de entrenamiento D . Una vez que el modelo ha sido entrenado puede ser utilizado con nuevas instancias, llamadas conjunto de pruebas. La capacidad de acertar correctamente a un concepto ante nuevas instancias diferentes a las del conjunto de entrenamiento se llama generalización. Los valores de las nuevas instancias pueden variar al conjunto de entrenamiento, por lo que la capacidad de generalización del modelo es importante. Mientras más se asemeje la distribución de los datos del conjunto de entrenamiento al real, el modelo dará mejores resultados.

Aplicaciones de aprendizaje en las que se provee la pareja: instancia con atributos y un valor objetivo ($\langle x, c(x) \rangle$), se conocen como problemas de aprendizaje *supervisado*. El ejemplo de determinar el tipo de animal, dónde a una instancia se tiene que asignar una categoría de un grupo finito es un problema de clasificación. Si se pretende asignar un valor continuo es una tarea de regresión, como predecir el precio de una casa en base a características y precios de otras casas. Por otro lado, si el conjunto de entrenamiento no contiene los valores objetivos se trata de problemas de aprendizaje no supervisado. El objetivo de este segundo tipo de aprendizaje es la agrupación.

Los algoritmos de aprendizaje maquina son herramientas claves para la minería de datos. Permiten encontrar regularidades implícitas en bases de datos. Han sido adoptadas en diversos sectores para descubrir patrones complejos de forma automática. Por ejemplo: en base a historiales crediticios generar modelos que permiten determinar si una nueva solicitud es rechazada o aceptada. Además proveen un marco de trabajo para realizar el aprendizaje de los datos.

3.1.1. Tarea de inducción: Sábados por la mañana adecuados para jugar tenis

Para ilustrar el funcionamiento de algunos de los algoritmos de aprendizaje, se utiliza como ejemplo la tarea que consiste en aprender el concepto: “Sábados por la mañana adecuados para jugar tenis” [34] representado por el atributo *JuegaTenis* con los valores 'Si' y 'No'. El cual se describe por cuatro atributos: *Cielo*, *Temperatura*, *Humedad* y *Viento*. Cada instancia corresponde a las condiciones registradas en diferentes Sábados y en el atributo *JuegaTenis* el valor 'Si', indica si fue un día adecuado para jugar y 'No' en caso contrario. Este último representa la clase.

La tarea consiste en construir un modelo que permita predecir, dadas las condiciones de un Sábado, si esté es adecuado para jugar tenis o no. Utilizando el conjunto de entrenamiento dada por la tabla 3.4. Adicionalmente se proporciona la siguiente instancia de prueba:

Día	Cielo	Temperatura	Humedad	Viento	JuegaTenis
D16	Nubes	Baja	Normal	Débil	?

Cuadro 3.3: Día en el que se desea predecir si es adecuado para jugar tenis.

3.2. Árboles de decisión

Es una familia de métodos de inducción que se caracteriza por representar el conocimiento en forma de árboles de decisión. Representación que resulta conveniente por su capacidad de expresión, ya que muestra de forma explícita las variables y sus relaciones. Un ejemplo de un árbol de decisión se muestra en la figura 3.2. Son métodos populares que han sido utilizados en

Día	Cielo	Temperatura	Humedad	Viento	JuegaTenis
D1	Sol	Alta	Alta	Débil	No
D2	Sol	Alta	Alta	Fuerte	No
D3	Nubes	Alta	Alta	Débil	Si
D4	Lluvia	Suave	Alta	Débil	Si
D5	Lluvia	Baja	Normal	Débil	Si
D6	Lluvia	Baja	Normal	Fuerte	No
D7	Nubes	Baja	Normal	Fuerte	Si
D8	Sol	Suave	Alta	Débil	No
D9	Sol	Baja	Normal	Débil	Si
D10	Lluvia	Suave	Normal	Débil	Si
D11	Sol	Suave	Normal	Fuerte	Si
D12	Nubes	Suave	Alta	Fuerte	Si
D13	Nubes	Alta	Normal	Débil	Si
D15	Lluvia	Suave	Alta	Fuerte	No

Cuadro 3.4: Ejemplos de entrenamiento para la clasificación del atributo *JuegaTenis*

una gran variedad de aplicaciones como: sistemas expertos, inteligencia de negocios, detección de fraudes, etc.

Estos procedimientos reciben como entrada un conjunto de instancias, donde cada instancia tiene asignada un valor discreto de un conjunto finito (clase). La construcción del árbol es de arriba hacia abajo guiado por la información que provee cada atributo. El resultado es un modelo con forma de árbol conformado por los atributos descriptivos. En donde las hojas (círculos) representan clases y el resto de los nodos (rectángulos) representan bloques de decisión sobre un atributo con una rama por cada posible valor de este. Para clasificar una instancia, se inicia en el nodo raíz, se evalúa dicho atributo y se selecciona la rama que corresponda al valor de la instancia dada. El proceso continua hasta que se alcanza un hoja, que determina la clasificación.

La estrategia general de estos métodos consiste en dividir los datos, de tal forma que se obtenga un conjunto de instancias donde todas tengan asociadas el mismo valor (clase). Para ilustrarla, suponga que el conjunto de entrenamiento consiste en puntos en un espacio de dos dimensiones, como lo muestra la figura 3.1. Los atributos consisten en los ejes x_1 y x_2 . Si se toma una línea que divida el plano sobre el eje x_2 sobre el punto w_{10} es posible observar que se obtiene una partición pura, es decir, contiene elementos de la misma clase (denotados por círculos). Las particiones puras implican que no es necesario continuar dividiendo los datos y representan una hoja. El proceso de división de los datos continua hasta que todas la particiones sean puras.

La elección del algoritmo depende principalmente del tipo de dato. ID3 es un procedimiento que se explica a continuación, que opera con datos discretos. Donde se tienen tantas ramas como valores toma un atributo. Esto es diferente si se usan valores continuos en donde

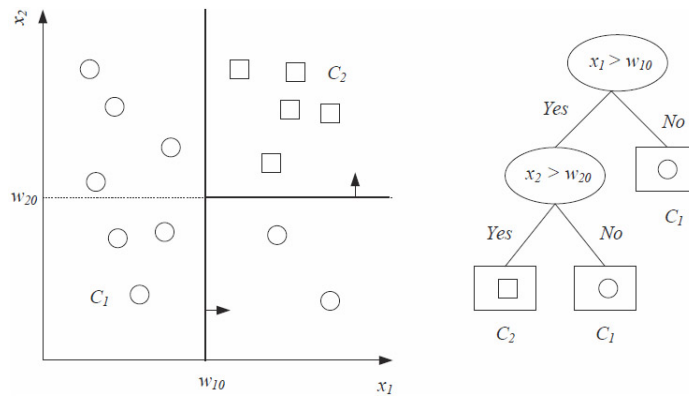


Figura 3.1: Ilustración de la partición por axis de un espacio de dos dimensiones.

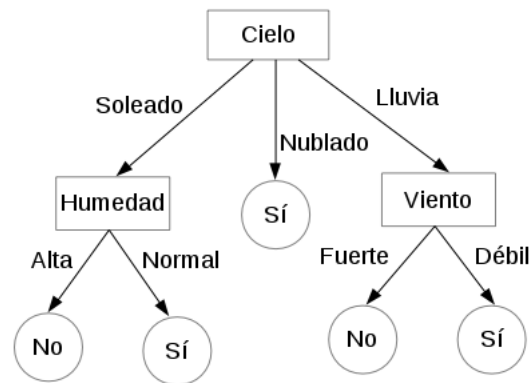


Figura 3.2: Árbol de decisión para la clasificación *JugarTennis*.

se puede tener una cantidad infinita de valores resultando en una cantidad infinita de ramificaciones, lo cual no es factible. Por lo que existen métodos que consideran esta situación, como es el caso de C4.5 [35], QUEST [27], que calculan un punto de división d . A pesar de sus diferencias estas metodologías comparten los siguientes pasos.

- Condición de paro.
- Criterio de selección.
- Literal de partición.

3.2.1. Algoritmo ID3

ID3 [34] es un algoritmo que permite crear árboles de decisión, que funciona con valores discretos. En la imagen 3.2 se muestra el árbol construido con esta metodología del ejemplo

jugar tenis. Para dar inicio a la construcción del árbol primero se debe decidir que atributo será utilizado para dividir los datos y ser la raíz. Cada atributo es evaluado mediante una prueba que determina que tan bien, por si solo, clasifica los ejemplos de entrenamiento. El atributo con mejores resultados es seleccionado para ser el nodo raíz. Se descarta el atributo y se generan subconjuntos de instancias por cada posible valor de dicho atributo. Lo que se representa como una ramificación en el árbol. En cada subconjunto se repite el proceso de selección y partición de datos, que termina cuando se han consumido todos los atributos. El detalle completo de ID3 se muestra en el algoritmo 1.

Algoritmo 1: ID3

Function ID3(*casos*, *atributo_objetivo*, *atributos*)

Data:

casos: Son los ejemplos de entrenamiento.

atributo_objetivo: Es el atributo cuyo valor corresponde a la clase.

atributos: Es una lista de atributos que puedan ser probados por el árbol de decisión aprendido.

Result: Árbol de decisión que clasifica correctamente los casos dados.

crea un nodo *Raiz* para el árbol

if *todos los casos son positivos* **then**

 | retorna el árbol solo con nodo *Raiz*, con la *etiqueta* = +

if *todos los casos son negativos* **then**

 | retorna el árbol solo con nodo *Raiz*, con la *etiqueta* = -

if *atributos está vacío* **then**

 | retorna el árbol con solo nodo *Raiz*, con la *etiqueta* = el valor más frecuente de atributo objetivo en casos.

A ← el atributo de *atributos* que mejor clasifique a *casos*

atributo decisión para *Raiz* ← *A*

for *cada posible valor v_i de A* **do**

 | agrega una nueva rama descendiente de *Raiz*, correspondiente v_i

 | sea *casos _{v_i}* un subconjunto de *casos* que tienen el valor v_i para *A*

 | **if** *casos _{v_i}* está vacío **then**

 | debajo de esta nueva rama, agrega un nodo hoja con la *etiqueta* = el valor más frecuente de *atributo_objetivo* en *casos*.

 | **else**

 | debajo de esta nueva rama agrega un subárbol

 | ID3(*casos _{v_i}* , *atributo_objetivo*, *atributos* - {*A*})

Criterio de selección: Ganancia de información

Una parte clave en el algoritmo ID3 es la selección de atributos. Es decir, que atributo será utilizado para separar los datos. Se busca elegir el atributo que produzca la partición

más pura. Para lo que hace uso de una propiedad llamada *ganancia de información*, que mide que tan bien un atributo, por si solo, separa las instancias de entrenamiento de acuerdo a la clase.

Para poder definir con precisión la ganancia de información, es necesario conocer una medida utilizada comúnmente en teoría de información, denominada *entropía de Shannon*, que caracteriza la impureza de una colección arbitraria de ejemplos. Por ejemplo, dada una colección S , la cual contiene instancias positivas y negativas de un concepto objetivo (clase), la entropía de S , relativa a una clasificación binaria esta dada por:

$$\text{Entropia}(S) \equiv -p_{\oplus} \log_2(p_{\oplus}) - p_{\ominus} \log_2(p_{\ominus})$$

Donde p_{\oplus} es la probabilidad de ejemplos positivos en S y p_{\ominus} es la probabilidad de ejemplos negativos en S . En todos los cálculos relativos a la entropía se define, $0 \cdot \log_2 0 = 0$.

Considere el ejemplo de jugar tenis, donde S es una colección de 14 ejemplos de una clasificación binaria, con 9 ejemplos positivos y 5 negativos (se adopta la notación $[9+, 5-]$ para representar la muestra de los datos). La entropía relativa a esta clasificación binaria es:

$$\text{Entropia}([9+, 5-]) = -(9/14) \log_2(9/14) - (5/14) \log_2(5/14) = 0.940$$

Note que la entropía es 0 si todos los miembros de S pertenecen a la misma clase. Por ejemplo, si todos los miembros son positivos $p_{\oplus} = 1$, entonces $p_{\ominus} = 0$ y se obtiene:

$$\text{Entropia}(S) = -1 \cdot \log_2(1) - 0 \cdot \log_2 0 = -1 \cdot 0 - 0 \cdot \log_2 0 = 0$$

Por otro lado, la entropía es 1 cuando la colección contiene el mismo número de ejemplos positivos y negativos, lo que implica que la entropía se mide entre 0 y 1.

Si la clasificación tiene c diferentes clases, entonces la entropía de S relativa a c clases se define como:

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

Donde p_i es la proporción de S que pertenece a la clase i . Note, que si el atributo seleccionado puede tomar c posibles valores, la entropía puede llegar a ser tan grande como $\log_2 c$.

Dada la entropía como medida de impureza en una colección de ejemplos de entrenamiento, es posible obtener la medida de eficacia de un atributo para clasificar los datos de entrenamiento, mediante la *ganancia de información*. Esta última simplemente es la reducción esperada en la entropía causada de separar los ejemplos acorde al atributo en cuestión. De forma más precisa, la ganancia de información, $\text{Ganancia}(S, A)$ de un atributo A , relativo a una colección de ejemplos S , se define como:

$$\text{Ganancia}(S, A) \equiv \text{Entropia}(S) - \sum_{v \in \text{Valores}(A)} \frac{|S_v|}{|S|} \text{Entropia}(S_v)$$

Donde $\text{Valores}(A)$ son todos los posibles valores que tiene el atributo A (o clases), y S_v es el subconjunto de S para el cual el atributo A tiene el valor v (esto es, $S_v = \{s \in S | A(s) = v\}$). Note que el primer termino en la ecuación es justo la entropía de la colección original S , y el segundo termino es valor esperado de la entropía después de separar S utilizando el atributo A . La entropía esperada descrita por el segundo termino es simplemente la suma de las entropías de cada subconjunto S_v , ponderado por la fracción de ejemplos $\frac{|S_v|}{|S|}$ que pertenecen a S_v . La $\text{Ganancia}(S, A)$ es por lo tanto la reducción esperada en la entropía originada por conocer el valor del atributo A .

Jugar Tenis

Para mostrar la forma en que se obtiene la ganancia de información, considere el ejemplo de 'Jugar Tenis' de la sección 3.1.1. En donde se tiene el atributo: *Viento*, que puede tomar los valores: *Debil* o *Fuerte*. Como anteriormente se mencionó, S es una colección con 14 ejemplos, [9+,5-]. De los cuales, 6 de los ejemplos positivos y 2 de los negativos tienen $\text{Viento}=\text{Debil}$, y el resto tiene $\text{Viento}=\text{Fuerte}$. La ganancia información debido a la separación de los 14 ejemplos originales por el atributo *Viento* puede ser calculada como:

$$\begin{aligned}
 \text{Valores}(\text{Viento}) &= \text{Debil}, \text{Fuerte} \\
 S &= [9+, 5-] \\
 S_{\text{Debil}} &\leftarrow [6+, 2-] \\
 S_{\text{Fuerte}} &\leftarrow [3+, 3-] \\
 \text{Ganancia}(S, \text{Viento}) &= \text{Entropia}(S) - \sum_{v \in \{\text{Debil}, \text{Fuerte}\}} \frac{|S_v|}{|S|} \text{Entropia}(S_v) \\
 &= \text{Entropia}(S) - (8/14)\text{Entropia}(S_{\text{Debil}}) - (6/14)\text{Entropia}(S_{\text{Fuerte}}) \\
 &= 0.940 - (8/14)0.811 - (6/14)1.00 \\
 &= 0.048
 \end{aligned}$$

La ganancia de información de los cuatro atributos es:

$$\begin{aligned}
 \text{Ganancia}(S, \text{Cielo}) &= 0.246 \\
 \text{Ganancia}(S, \text{Humedad}) &= 0.151 \\
 \text{Ganancia}(S, \text{Viento}) &= 0.048 \\
 \text{Ganancia}(S, \text{Temperatura}) &= 0.029
 \end{aligned}$$

El atributo que genere la mayor ganancia será utilizado para separar el conjunto de instancias. En el ejemplo, el atributo con mayor ganancia de información es *Cielo* el cual corresponde a la raíz del árbol. Las instancias serán divididas acorde a los valores de dicho atributo y por cada subconjunto crecerá un árbol siguiendo el mismo proceso. El árbol de

decisión aprendido por ID3 de los 14 ejemplos de entrenamiento de la tabla 3.4 se muestra en la figura 3.2.

Una de las ventajas de los árboles de decisión creados por ID3, es su capacidad de generalización. Donde los atributos en los niveles superiores, más cercanos al nodo raíz, son los atributos que describen las clases de manera general. Desde un punto de vista de análisis, puede ser útil para encontrar algún elemento que revele alguna intuición de los datos, mediante la relación jerárquica de los atributos. Otra característica adicional de ID3, es que permite descartar aquellos atributos que no aportan valor, esto sucede si se crea un nodo hoja sin necesidad de utilizarlos; esto ubica a este algoritmo como un procedimiento comúnmente utilizado para la reducción de datos, que permite obtener un conjunto de atributos relevantes.

Sobreajuste

El algoritmo ID3 crece cada rama del árbol lo suficiente para clasificar perfectamente los ejemplos de entrenamiento, lo cuál parece una estrategia razonable. No obstante, puede presentar dificultades cuando los datos son ruidosos o se cuenta con pocos casos de entrenamiento. En cualquiera de los dos casos, se presenta un *sobreajuste*. Esto último consiste en que el modelo producido es muy particular a los casos de entrenamiento y pierde su capacidad de generalizar. Por lo que ante nuevos casos, los resultados de clasificación tienden a ser malos.

Definición 3.1. Dado un espacio de hipótesis H , una hipótesis $h \in H$ se dice se sobreajusta a los datos de entrenamiento si existe alguna alguna hipótesis alternativa $h' \in H$, tal que esa hipótesis h contiene un error menor que h' sobre los ejemplos de entrenamiento, pero h' tiene un error menor que h sobre la distribución entera de instancias.

Cuando el conjunto de entrenamiento contiene errores o ruido (valores con alta variabilidad). El algoritmo lo resuelve generando más nodos, agregando profundidad; produciendo un árbol más complejo y difícil de interpretar. Donde se pierde una de las cualidades más atractivas de estos modelos, una estructura representativa de fácil compresión. Para mostrar como sucede el sobreajuste ante ruido o errores, considere agregar el siguiente caso, incorrectamente etiquetado como negativo, a los ejemplos definidos en la tabla 3.4.

Día	Cielo	Temperatura	Humedad	Viento	JuegaTenis
D17	Sol	Alta	Normal	Fuerte	No

El árbol generado considerando los ejemplos originales es el ilustrado en la figura 3.2. Sin embargo, al considerar este ejemplo, ID3 construirá un modelo más complejo. En particular el nuevo caso será colocado en la segunda hoja de la rama izquierda del árbol de la figura 3.2, entre los casos positivos $D9$ y $D11$. Debido a que el nuevo caso es etiquetado como negativo, ID3 buscará un nuevo atributo como nodo de decisión para separar este nuevo ejemplo de los dos anteriores. El resultado será un árbol de decisión (h) más complejo que el original de la figura 3.2 (h'), en donde h se ajustará perfectamente a los casos de entrenamiento y h' no.

El nuevo nodo es consecuencia del ajuste a la instancia con ruido. El cuál es un problema de aprendizaje, que concierne a los árboles de decisión y otros métodos.

Existen diferentes alternativas para evitar el sobreajuste en procedimientos de árboles. Las cuales pueden ser agrupadas en dos tipos distintos:

- Detener el crecimiento de manera temprana, antes que alcance el punto donde clasifique perfectamente los datos de entrenamiento.
- Permitir al árbol sobreajustarse a los datos y posterior podar el árbol.

Siendo el segundo punto el que ha presentado mejores resultados en la práctica. La poda consiste en convertir un subárbol en un nodo hoja, la hoja es la clase moda de los registros de entrenamiento asociados a ese subárbol. Para llevar a cabo esta técnica, se requiere un conjunto adicional, al de entrenamiento y de pruebas, para propósitos de validación. El subárbol es eliminado solo si el árbol podado resultante no se comporta peor con el conjunto de validación. Por lo que cada rama, originada por irregularidades en el conjunto de entrenamiento, muy probablemente sea podada, debido a que, es poco probable que las mismas condiciones se presenten en el conjunto de validación. Los nodos son podados iterativamente, siempre seleccionando el nodo cuya eliminación mejore la precisión del árbol, sobre el conjunto de validación. La poda de árboles continua hasta que sea perjudicial.

3.2.2. Bosques Aleatorios

Hoy en día es cada vez más común que las bases de datos contengan una gran cantidad de registros y características. Lo que complican la tarea de análisis, donde de primera instancia ya no resulta tan trivial encontrar la relación entre una gran cantidad de ambas. Dicho incremento, también impacta en la cantidad de errores durante su captura así como la cantidad de atributos no significativos, lo que resulta en una mayor variabilidad en los datos. Uno de los procedimientos que se ve afectado por esta variabilidad, como se mencionó anteriormente son los árboles de decisión en los que ocasiona un sobreajuste, lo que resulta en modelos sesgados con poca capacidad de generalización y pobre desempeño de clasificación.

Una técnica que permite reducir la varianza estimada de una función de predicción, es *embolsado (bagging)*[4]. Que consiste en crear K árboles de decisión, cuando se tiene una nueva instancia, cada árbol la evalúa y emite un voto con alguna de las clases definidas, la clasificación final es aquella con mayores votos. En embolsado, para hacer crecer los árboles, se crean N muestras con reemplazo de tamaño K con instancias del conjunto de entrenamiento, tomadas de forma uniforme aleatoria. Por lo que se espera existan repetidos, ya que la muestra es creada con reemplazo. El objetivo de realizar estas muestras, es reducir el sesgo que ocasiona el ruido al promediar los modelos y por lo tanto reducir la varianza. Al distribuir las instancias es poco probable el ruido se distribuya de manera uniforme en todas las muestras. Por lo que cada árbol crecerá de forma distinta, donde esta diversidad permite reducir el impacto que tienen aquellos árboles sesgados.

Por otro lado *Bosques Aleatorios* [5] es una modificación de *embolsado*. En la que los vectores utilizados para construir el árbol contienen una selección arbitraria de características,

dicha configuración permite mejorar la precisión. El resultado son árboles de-corelacionados. Aplicaciones como puede ser diagnóstico médico o recuperación de texto, presentan una gran cantidad de atributos, entre miles o cientos de miles, en donde cada atributo aporta poca información. Optar por construir un único árbol de decisión y clasificar en base a este resultaría un poco mejor que una selección aleatoria de una clase. Además para cuestiones de análisis dicho árbol sería incomprensible. Esta selección arbitraria de atributos utilizada por los bosques aleatorios, ha presentado mejoras en los niveles de precisión. Siendo también más fácil interpretar un grupo de árboles de decisión sencillos con un grado aceptable de generalización. Es posible definir a los bosques aleatorios como:

Definición 3.2. Un bosque aleatorio es un clasificador que consiste de una colección de clasificadores de árboles $h(x, \Theta_k, k = 1, \dots)$ donde Θ_k son vectores aleatorios independientes idénticamente distribuidos y cada árbol emite un voto por la clase más probable de la entrada x .

Un bosque aleatorio sencillo, puede ser seleccionar instancias de forma aleatoria. A partir de cada muestra, hacer crecer cada árbol puede crecer usando la metodología de ID3. En el algoritmo 2 se muestra el detalle del procedimiento utilizando ID3 para construir los árboles.

Algoritmo 2: Bosques Aleatorios para Clasificación

Result: Conjunto de árboles: $\{T_b\}_1^B$

for $b \leftarrow 1$ **to** B **do**

(a) Generar una muestra Z^* de tamaño N del conjunto de entrenamiento.

(b) Crear un árbol T_b utilizando la metodología ID3.

end for

Para realizar clasificación de un nuevo punto x :

Clasifica: Sea $\hat{C}_b(x)$ sea la predicción de b -ésimo árbol dentro del bosque. Entonces

$\hat{C}_{final}(x) = \text{voto mayoritario } \{\hat{C}_b(x)\}_1^B$

3.3. Bayes Ingenuo

Dentro del aprendizaje maquina existe un enfoque probabilístico para realizar la inferencia de conceptos, que consiste en construir un modelo que estima la probabilidad de pertenecer a una clase. Provee un enfoque más flexible que los modelos de árboles que determinan la clasificación en base a decisiones excluyentes sobre los atributos descriptivos. En donde si una instancia no es consistente con el conjunto de entrenamiento se puede obtener el resultado incorrecto. Por ejemplo, suponga que se tiene una instancia de la clase B que provee un valor en el atributo a_x que esta asociado a un nodo terminal de la clase A ; un modelo de árbol elegiría esta última clase, sin considerar la información que aportan el resto de los atributos.

Es un enfoque que resulta de interés para la minería de datos por diversas razones. Una de ellas es que la probabilidad que estima indica de forma cuantitativa que tan parecida es

una instancia a una clase. Lo cual puede ser utilizado para refinar la toma de decisiones. Por ejemplo, los clientes cuya probabilidad de pertenecer a la categoría 'leal' sea mayor a un 80 % ofrecer descuentos especiales. A diferencia de los árboles de decisión que únicamente proveen de una clase explícita.

La estrategia de clasificación de estos modelos es simple, asignar la instancia a la clase con mayor probabilidad. Las tareas de clasificación asignan la categoría a una nueva instancia con base a la información disponible, donde se busca seleccionar la clase más parecida, es decir; la clase más probable.

El cálculo de las probabilidades está en términos del conjunto de entrenamiento. Está condición implica que se trata de un problema de probabilidad condicional. Para calcular dicha probabilidad se hace uso del teorema de Bayes. Comúnmente los procedimientos que utilizan el teorema se conocen como métodos bayesianos. Un ejemplo de este tipo de métodos es Bayes Ingenuo, el cuál es sencillo de implementar y se ha reportando que es competitivo con otros más complejos como SVM y redes neuronales.

3.3.1. Teorema de Bayes

Estimar las probabilidad de una clase dado el conjunto de entrenamiento es posible mediante el teorema de Bayes. El cuál provee un método directo basado en: datos existentes, probabilidades *a priori* de las clases y conjunto de entrenamiento D , así como la probabilidad condicional del conjunto D dada la clase. Donde *A priori* se refiere a las probabilidades sin contar factores externos, por ejemplo: la probabilidad de una clase sin contar los atributos descriptivos.

Antes de definir el teorema de Bayes, es necesario presentar la siguiente notación. Sea $P(c)$ la probabilidad de la clase c , comúnmente llamada probabilidad *a priori* de c . De forma similar, se utiliza a $P(D)$ para denotar la probabilidad *a priori* del conjunto de entrenamiento D que será observado, esto es, la probabilidad de D sin considerar la clase a la que pertenece. Siguiendo, $P(D|c)$ denotará la probabilidad de los datos observados D dada la clase. De manera general, se indica $P(x|y)$ para denotar la probabilidad de x dado y . En problemas de clasificación se desea calcular $P(c|D)$ que representa la probabilidad de la clase dados los datos de entrenamiento. $P(c|D)$ se conoce como la probabilidad *a posteriori*, que refleja la influencia del conjunto de entrenamiento D . En contraste con la probabilidad *a priori* $P(c)$ la cual es independiente de D . El teorema de Bayes provee una forma de calcular la probabilidad *a posteriori* $P(c|D)$ a partir de las probabilidades: $P(c)$, $P(D)$ y $P(D|c)$. El teorema de Bayes está dado por la siguiente formula:

$$P(c|D) = \frac{P(D|c)P(c)}{P(D)} \quad (3.1)$$

3.3.2. Descripción de Bayes Ingenuo

Un método de clasificación probabilística es Bayes Ingenuo. Su enfoque consiste en, dada una nueva instancia, estimar la clase más probable dado el conjunto de entrenamiento. Para

esto, se calculan las probabilidades condicionales de cada clase mediante el teorema de Bayes y se devuelve aquella con la máxima probabilidad como se muestra en la función c_{MAP} . Siendo esta el resultado del clasificador.

$$\begin{aligned}
 c_{MAP} &\equiv \arg \max_{c \in C} P(c|D) \\
 &= \arg \max_{c \in C} \frac{P(D|c)P(c)}{P(D)} \\
 &= \arg \max_{c \in C} P(D|c)P(c)
 \end{aligned} \tag{3.2}$$

Note que en la ecuación anterior desaparece el término $P(D)$ ya que se asume equiprobable y no afecta a $\arg \max$ [30].

De forma más específica recibe como entrada un conjunto de instancias $\langle x, v_i \rangle$ donde x se representa un vector con los valores de los atributos descriptivos $\langle a_1, a_2, \dots, a_n \rangle$ y v_i corresponde al concepto objetivo o clase. Ante una nueva instancia se solicita al procedimiento asignarle una clase. Es decir, calcular la clase más probable c_{MAP} en base a los atributos descriptivos dentro del conjunto de entrenamiento D . Con base a esta información la fórmula 3.2 se puede expresar como:

$$c_{MAP} = \arg \max_{c_j \in C} P(c_j | a_1, a_2, \dots, a_n)$$

Al aplicar el teorema de Bayes se obtiene la expresión:

$$\begin{aligned}
 &= \arg \max_{v_j \in V} \frac{P(a_1, a_2, \dots, a_n | v_j) P(v_j)}{P(a_1, a_2, \dots, a_n)} \\
 &= \arg \max_{v_j \in V} P(a_1, a_2, \dots, a_n | v_j) P(v_j)
 \end{aligned} \tag{3.3}$$

Los dos términos de la ecuación 3.3 se estiman utilizando el conjunto de entrenamiento. El segundo término $P(v_j)$ es el más sencillo de calcular. Simplemente se requiere contar la frecuencia con la que cada clase v_j ocurre dentro de D . El segundo resulta un poco más complejo ya que se tendrían que calcular una gran cantidad de probabilidades. Sin embargo, el clasificador Bayes Ingenuo se basa en la suposición ingenua que los atributos son condicionalmente independientes. Razón por la que lleva ese nombre. Por lo que la probabilidad de la conjunción a_1, \dots, a_n es el producto de las probabilidades de los atributos de forma individual: $P(a_1, a_2, \dots, a_n | v_j) = \prod_i^n P(a_i | v_j)$. Sustituyendo esto en la ecuación 3.3 se obtiene el enfoque utilizado por el clasificador Bayes Ingenuo.

$$V_{NB} = \arg \max_{v_j \in V} P(v_j) \prod_i^n P(a_i | v_j) \tag{3.4}$$

Donde V_{NB} denota la clase con la probabilidad más alta, calculada por el clasificador Bayes Ingenuo en base al conjunto de entrenamiento. Para esto es necesario calcular la probabilidad

de cada atributo dada la clase, $P(a_i|v_i)$, lo que implica que la cantidad de probabilidades a calcular será igual a la cantidad de atributos por la cantidad de clases.

Jugar Tenis

Nuevamente retomando el ejemplo ilustrativo de 'Jugar Tenis' de la sección 3.1.1. Donde se desea predecir si un sábado es adecuado para jugar tenis en base al conjunto de entrenamiento. El cuál está conformado por 14 ejemplos que se indican en la tabla 3.4 como se vio en la sección 3.1.1. El concepto objetivo se encuentra descrito por los atributos: *Cielo*, *Temperatura*, *Humedad* y *Viento*. En esta ocasión se utilizará el clasificador Bayes Ingenuo para clasificar el sábado representado por la siguiente instancia:

Día	Cielo	Temperatura	Humedad	Viento	JuegaTenis
D18	Sol	Baja	Alta	Fuerte	?

El objetivo es asignar uno de los valores del concepto objetivo: "Si" o "No", de la clase *JugarTenis*. Es posible predecir dicho valor, mediante el clasificador Bayes ingenuo utilizando la ecuación 3.4.

$$\begin{aligned}
 N_{NB} &= \arg \max_{v_j \in \{si, no\}} P(v_j) \prod_i P(a_i|v_i) \\
 &= \arg \max_{v_j \in \{si, no\}} P(v_j) P(\text{Cielo} = \text{Sol}|v_j) P(\text{Temperatura} = \text{Baja}|v_j) \\
 &\quad P(\text{Humedad} = \text{Alta}|v_j) P(\text{Viento} = \text{Fuerte}|v_j)
 \end{aligned} \tag{3.5}$$

En la ecuación anterior se sustituyó a_i por los 4 atributos del ejemplo, dejándose indicado v_j que tomará los valores de cada clase. Por lo que para obtener N_{NB} se requieren calcular 10 probabilidades; 2 por cada clase denotadas por $P(v_j)$ y 8 correspondientes a la probabilidad condicional $P(a_i|v_j)$. Las probabilidades *a priori* de las clases pueden ser calculadas con base en la frecuencia de estas sobre los 14 ejemplos.

$$P(\text{JuegaTenis} = si) = 9/14 = 0.64$$

$$P(\text{JuegaTenis} = no) = 5/14 = 0.36$$

De forma similar, es posible estimar las probabilidades condicionales para cada atributo. Por ejemplo, si *viento=fuerte* se obtendría:

$$P(\text{viento} = \text{fuerte} | \text{JuegaTenis} = si) = 3/9 = 0.33$$

$$P(\text{viento} = \text{fuerte} | \text{JuegaTenis} = no) = 3/5 = 0.60$$

Utilizando estas y calculando de la misma forma las probabilidades de los demás atributos y utilizando la ecuación 3.5 para obtener V_{NB} como sigue (se omiten los nombres de los atributos para ser más breve):

$$P(si)P(soleado|si)P(frio|si)P(alto|si)P(fuerte|si) = 0.0053$$

$$P(no)P(soleado|no)P(frio|no)P(alto|no)P(fuerte|no) = 0.0206$$

Recordando que la regla de clasificación de Bayes Ingenuo es estimar la clase con la máxima probabilidad. En este ejemplo el valor más alto está asociado a la clase *no*, por lo que el resultado de la clasificación inferida por este método es $JugarTenis = no$.

Clasificación de documentos de texto

La clasificación de documentos de texto es una aplicación importante dentro de la minería de datos. Por ejemplo, en la internet cada día se genera una gran cantidad de textos como pueden ser: páginas web, correos electrónicos, publicaciones de redes sociales, etc. La clasificación de correos electrónicos en: “legítimo” y “basura” con base a la preferencia de usuarios, es un aplicación práctica. La cuál, se ofrece como servicio por diferentes motores de correos.

Para realizar clasificación de textos uno de los primeros inconvenientes a resolver es expresar un documento en el formato requerido por los procedimientos. Los cuáles, como se ha mencionado anteriormente, reciben como entrada tuplas de la forma $\langle x, y \rangle$, donde x representa un vector con valores $\langle x_1, x_2, \dots, x_n \rangle$. Dos enfoques para representar un documento son: el primero se conoce como Bolsa de palabras, en donde cada atributo corresponde a una palabra y su valor a la ocurrencia de la palabra en el documento. Dicha representación se basa en vocabularios, un ejemplo de vocabulario puede las palabras del idioma Inglés, que consta de aproximadamente 500,000 palabras diferentes. Para este caso, una alternativa más eficiente consiste en generar un vocabulario con las palabras distintas existentes en el conjunto de entrenamiento, también conocido como *corpus* dentro de un contexto de análisis de textos. Suponiendo que se haya generado un vocabulario de 1000 palabras, la representación del documento sería: $\langle (a_1, a_2, \dots, a_{1000}), clase \rangle$. El segundo enfoque de representación consiste en la tupla $\langle d, y \rangle$, donde d representa el texto del documento. Una de las ventajas de este último sobre el primero, es que algunas palabras son muy poco frecuentes por lo que al final se tiene un vector con una gran cantidad de ceros que no aportan información ocupando espacio de memoria. Esta última opción es la utilizada por el clasificador Bayes Ingenuo para la clasificación de texto.

Clasificación de documentos con Bayes Ingenuo

Dado el *corpus* de entrenamiento compuesto por documentos etiquetados con su respectiva clase y representados de la forma $\langle d, y \rangle$ se requiere construir un modelo mediante el clasificador Bayes Ingenuo que ante un nuevo documento sea capaz de predecir la categoría a la que pertenece. Esto es, obtener la clase con la probabilidad más alta dado por V_{NB} , dónde

la estimación de éstas estará en términos de la ocurrencia de las palabras en el *corpus* de entrenamiento.

Para obtener la V_{NB} de la ecuación 3.4, se requiere estimar las probabilidades de los términos $P(v_j)$ y $P(a_i|v_j)$. La primera se obtiene con base a la ocurrencia de cada clase en el conjunto de entrenamiento. Como se indica en la fórmula 3.6 donde n_c es la cantidad de documentos de la clase c y n es el total de documentos de entrenamiento.

$$P(c) = \frac{n_c}{n} \quad (3.6)$$

Por otro lado para obtener $P(a_i|v_i)$ se requiere estimar las probabilidades condicionales de cada palabra dentro *corpus* por clase. Para lo que se asume independencia entre variables, lo que significa que es igualmente probable encontrar cualquier palabra en una posición arbitraria. Por ejemplo, bajo este supuesto de independencia en el presente capítulo se podría encontrar la palabra “probabilidad” después de la palabra ”aprendizaje”; sin embargo, esta suposición no es correcta dado que sabemos que es más probable que la palabra que procede a ”aprendizaje” sea ”maquinal”. A pesar de ello en la práctica el clasificador Bayes Ingenuo presenta buenos resultados. La cantidad de probabilidades que se tiene que calcular corresponde a: la cantidad de palabras en el vocabulario (v) por la cantidad de clases (c). Esto es, si v es de tamaño 1000 y se tiene dos clases, se tendrían que estimar 1000x2 probabilidades condicionales.

Es posible estimar $P(a_i|v_j)$ utilizando la fórmula 3.6. Con base a la cantidad de veces que ocurre la palabra a_i en los documentos de la clase v_j . Sin embargo, si la palabra a_i no ocurre en la clase v_j se obtendría una probabilidad de cero, este valor repercutiría sobre los cálculos del clasificador Bayes Ingenuo, ya que para obtener el resultado se multiplican todas las probabilidades. Lo que implica que ante la presencia de un cero el resultado final es cero. Para evitar esta dificultad se utiliza el estimador *Laplace smoothing*[30] que agrega un uno a cada conteo.

$$P(a_i|v_j) = \frac{n_k + 1}{n + |\text{vocabulario}|} \quad (3.7)$$

donde n es la cantidad total de ocurrencias de todas las palabras existentes en los documentos de entrenamiento cuya clase es v_j , n_k es el número de veces que la a_i palabra ocurre en el documento que se desea clasificar y $|\text{vocabulario}|$ es el total de palabras distintas en el conjunto de entrenamiento independiente de la clase.

El detalle del procedimiento Bayes Ingenuo para clasificar texto se muestra en el algoritmo 3[30].

3.4. Vecinos más cercanos

Los métodos de aprendizaje basados en instancias son uno de los enfoques más fáciles para realizar clasificación. Estos almacenan en memoria las instancias de entrenamiento y para realizar la clasificación de una nueva instancia buscan las más parecidas y con base al

Algoritmo 3: Procedimiento clasificador Bayes Ingenuo para clasificar texto.

```

Function EntrenaBayesIngenuo( $C, D$ )
1  |  $V \leftarrow \text{ExtraeVocabulario}(D)$ 
2  |  $N \leftarrow \text{CuentaDocumentos}(D)$ 
3  | for  $c \in C$  do
4  |   |  $N_c \leftarrow \text{ContarDocumentosEnClase}(D, c)$ 
5  |   |  $\text{prior}[c] \leftarrow \frac{N_c}{N}$ 
6  |   |  $\text{texto}_c \leftarrow \text{ConcatenarTextoTodosDocsEnClase}(D, c)$ 
7  |   | for  $t \in T$  do
8  |   |   |  $T_{ct} \leftarrow \text{CuentaTokens}(\text{texto}_t, t)$ 
9  |   |   | end for
10 |   | for  $t \in T$  do
11 |   |   |  $\text{condprob}[t][c] \leftarrow \frac{T_{ct}+1}{\sum_{t'} (T_{ct'}+1)}$ 
12 |   |   | end for
13 |   | return  $V, \text{prior}, \text{condprob}$ 
14 |   | end for
15 | end
16 | Function ClasificaBayesIngenuo( $C, V, \text{prior}, \text{condprob}, d$ )
17 |   |  $W \leftarrow \text{ExtraeTokensDeDocs}(V, d)$ 
18 |   | for  $c \in C$  do
19 |   |   |  $\text{probttotal}[c] \leftarrow \log \text{prior}[c]$ 
20 |   |   | for  $t \in W$  do
21 |   |   |   |  $\text{probttotal}[c] += \log \text{condprob}[t][c]$ 
22 |   |   |   | end for
23 |   |   | end for
24 |   | return  $\text{argmax}_{c \in C} \text{probttotal}[c]$ 
25 | end

```

resultado de dicha consulta se determina la clase. El método más sencillo es el clasificador *Rote*[42], el cuál mantiene el conjunto de entrenamiento en memoria, donde se podría decir que lo memoriza. El cuál para realizar la clasificación de una nueva instancia busca su coincidencia exacta con al menos una dentro del conjunto de entrenamiento, si dicha búsqueda es positiva se asigna la clase asociada; en caso contrario no se asigna ningún valor. Un inconveniente evidente de este método es que habrá varios registros que no puedan ser etiquetados debido a que es poco probable estén en memoria.

Otro método de este tipo es el de Vecinos más cercanos (k NN)[11] [42]. Que para llevar a cabo la clasificación de una instancia, consulta las k más cercanas obteniendo sus vecinos, con base a estos determina la clase moda y se la asigna. De forma general, el enfoque de k NN es buscar instancias similares a la que se desea etiquetar y asignar la clase predominante. Donde las instancias más parecidas indican de manera más fiable la categoría.

El método de k NN requiere tres parámetros: conjunto de entrenamiento a almacenar, una medida de similitud para comparar las instancias y la cantidad de k de vecinos que se consultará. El procedimiento asume que las instancias representan puntos en un espacio de n dimensiones \mathbb{R}^n . Dado un punto p que se desea clasificar, busca los k vecinos más cercanos en base a una métrica de distancia o similitud. Para encontrar los k puntos más cercanos se calcula la distancia del punto p contra todos los puntos del conjunto de entrenamiento. La clasificación se realiza en base al conjunto de vecinos, donde cada uno emite un voto con su respectiva clase, aquella con mayor cantidad de votos será asignada al punto p .

Una de las métricas de similitud más utilizadas es la distancia euclidiana. Por lo que el método permite operar con valores continuos. Si se cuenta con valores discretos del tipo: “Alto”, “Bajo”; se puede realizar un procesamiento previo para reemplazar la etiqueta con un código numérico como: 1 y 0 respectivamente. La distancia euclidiana entre dos puntos: $p(p_1, p_2, \dots, p_n)$ y $q(q_1, q_2, \dots, q_n)$ esta dada por:

$$\sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \dots + (p_n - q_n)^2} \quad (3.8)$$

Para ilustrar el funcionamiento de k NN, considere el ejemplo en el que las instancias son puntos en un espacio de dos dimensiones y la clase tiene dos posibles valores: positivos y negativos, denotados por + y - respectivamente. En la figura 3.3 [42] se indica el punto p con una \times del que se desea obtener sus vecinos para determinar su clasificación. Si se considera únicamente a un vecino con $k=1$, el algoritmo clasifica a q como negativo. Sin embargo, si se consideran 3 vecinos más cercanos asignando $k = 3$ la clasificación es un ejemplo positivo. El detalle del procedimiento se muestra en el algoritmo 4.

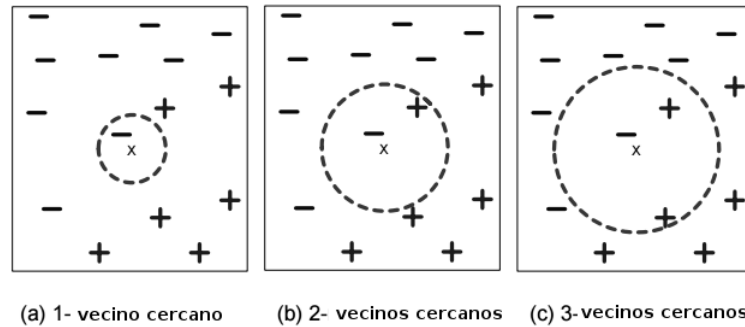


Figura 3.3: Vecinos más cercanos. Ejemplo de casos positivos y negativos con un punto x_q para clasificar.

Algoritmo 4: Vecinos más cercanos

Data: D , k , objeto prueba $z = (x, y')$

Result: y'

Calcular la distancia d entre z y cada objeto $(x, y) \in D$.

Seleccionar $D_z \subseteq D$, donde D_z es la lista k objetos más cercanos a z dentro del grupo de entrenamiento a z .

Calcular $y' = \operatorname{argmax}_v \sum_{(x_i, y_i) \in D_z} I(v = y_i)$

Donde: v es la etiqueta de la clase, y_i es la clase para el i -ésimo vecino más cercano e $I(\cdot)$ es una función indicador que retorna 1 si el argumento es verdadero 0 en caso contrario.

Una de las características del k NN es que el modelo se conforma por el conjunto de entrenamiento completo. El procedimiento se limita a almacenar las instancias en la etapa de entrenamiento. A diferencia de otros métodos que construyen un modelo explícito y con base a este realizan la clasificación, como Bayes Ingenuo. Debido a esta característica los métodos basados en instancias se conocen como perezosos.

Por lo que la etapa de clasificación puede llegar a ser costosa debido a la cantidad de distancias a calcular para obtener los vecinos. Tarea que se complica más cuando el número de características e instancias es alto. Para mejorar la eficiencia de k NN en la etapa de clasificación se han desarrollado diferentes técnicas. Una de ellas consiste en indexar los casos de entrenamiento, con el propósito de identificar los vecinos más cercanos de forma más eficiente, por un costo adicional de memoria. Un método para indexar es *kd-tree* (Bentley 1975; Friedman 1977), que consiste en una estructura de árbol donde las instancias son nodos en el árbol. Cuando se consultan los vecinos del punto p se recorre el árbol hasta llegar a un nodo y se devuelven los puntos almacenados.

Sin embargo, k NN es un clasificador conveniente en situaciones donde las instancias no son linealmente separables. Son una alternativa a otros métodos que realizan particiones

sobre los datos cómo árboles de decisión. Es un enfoque más sencillo a otros métodos más complejos como máquinas de soporte vectorial[14] que requieren proyectar a otro espacio los puntos para realizar una partición.

Consideraciones

Son diferentes los factores que pueden afectar el desempeño de k NN. El primero es el valor de k . Si k es muy pequeño podría no aportar suficiente información para seleccionar la clase. Por otro lado si k es muy grande la vecindad podría incluir muchos puntos de otras clases. Resultado en un conjunto de vecinos con una distribución similar de clases.

La elección de la clase es otro factor importante. La clase mayoritaria podría resultar no tan efectiva. Por ejemplo, cuando los puntos más distantes asociados a una misma clase predominan sobre, aquellos más cercanos asociados a otra. Bajo la premisa que los puntos más cercanos indican de forma más precisa la clase; una mejora consiste en asignarle un peso a la votación de cada vecino. Donde el peso del voto depende de su distancia al punto p , el voto de los puntos más cercanos tendrá mayor relevancia. La clase de los puntos más cercanos será de mayor consideración que la de aquellos más distantes. El peso del voto se puede calcular utilizando la siguiente fórmula:

$$y' = \operatorname{argmax}_v \sum_{(x_i, y_i) \in D_z} w_i I(v = y_i) \quad (3.9)$$

donde

$$w_i \equiv \frac{1}{d(x_q, x_i)^2} \quad (3.10)$$

Otro factor es la elección de la métrica de distancia. Existen diferentes medidas sin embargo se debe seleccionar aquella que indique que mientras menor sea la distancia entre dos puntos mayor es el parentesco entre ellos. El cálculo de las distancias considera todos los atributos a diferencia de otros métodos, como es el caso de árboles de decisión que descarta atributos que no aportan información. Lo que implica que las distancias pueden verse dominadas por uno de los atributos. Por ejemplo, considere una base de datos con información de empleados; donde la altura de una persona varía entre 1.5-1.9 metros, el peso entre 50kg-70kg y se tiene un ingreso mensual con un rango de \$10,000 a \$1,000,000. En este ejemplo, dos personas con altura y peso similar pero con un salario de \$10,000 y \$1,000,000 quedarían a una distancia muy separada, por lo que se descartarían como vecinos a pesar de la similitud entre los otros dos atributos. Un enfoque para sobrellevar este problema es normalizar los valores de las instancias, para que un atributo cuyos valores tengan un rango amplio no tenga más peso que los demás.

Adicionalmente, estas métricas se ven afectadas por las instancias con muchas dimensiones. Esto es, pueden generar distancias que carecen de sentido. Por ejemplo, se obtienen distancias pequeñas para puntos distantes y viceversa. Lo que se conoce como la maldición

de las dimensiones. La métrica de distancia Euclidiana es muy sensible a altas dimensiones. Si la aplicación lo permite una opción es descartar dimensiones seleccionando aquellas más relevantes para la tarea de clasificación, como en la clasificación de texto [49].

3.5. k-Medias

La tarea de análisis de grupos permite colocar objetos similares en grupos. La idea es que los miembros de un grupo tengan características similares entre ellos y difieran de los miembros de otros grupos. Mientras más evidente sea la diferencia entre grupos, mejor es la agrupación. Por ejemplo, separar los estudiantes de un curso en dos grupos, la opción más evidente es por género, el resultado sería un grupo con miembros del género femenino y el otro con miembros del género masculino. Los grupos se identifican mediante un representante, el cuál puede ser visto como una abstracción que encapsula las características comunes a todos sus miembros.

El análisis de grupos tiene diversas aplicaciones. Una de ellas es la segmentación, por ejemplo en Biología, agrupar especies de acuerdo a sus características es útil para definir taxonomías. Otra aplicación es la detección de anomalías, donde aquellos objetos aislados que no sean similares a ninguno de los miembros de un grupo son de interés para una inspección; en un sector financiero una transacción atípica puede representar un fraude. Por otro lado, la agrupación es utilizada para propósitos de compresión; por ejemplo, en tareas de transmisión de imágenes donde es posible agrupar los píxeles cercanos y enviar al representante. Esta última de aplicación es útil cuando muchos objetos son similares y es aceptable perder un poco de información para reducir la cantidad de datos.

3.5.1. Algoritmo K -medias

k -Medias[26] es un algoritmo de agrupación que permite separar objetos en k grupos. De forma similar a k NN, asume que cada instancia es un punto en un espacio de n dimensiones. Su enfoque consiste en asignar un punto p al grupo cuyo representante, denominado centroide, sea más parecido.

El procedimiento de k -Medias inicia seleccionando k puntos en \mathbb{R}^n de forma aleatoria para ser los centros de grupos iniciales. Posteriormente se iteran los siguientes pasos hasta que no existan cambios en los centroides.

1. **Asignación de grupo:** Se calcula la distancia de cada punto contenido en D a cada centroide y se asigna al grupo del centroide más cercano.
2. **Cálculo de centroide:** Una vez que se han asignado todos los puntos de entrenamiento a cada grupo. Cada centroide es actualizado, el nuevo valor se obtiene calculando el centro de todos los puntos contenidos en el grupo.

El método de asignación a un grupos utilizado por k -Medias se expresa en la fórmula 3.11. Que devuelve la distancia mínima entre un punto $p \in D$ a cada centroide. La distancia

Euclidiana es la medida por defecto utilizada por el procedimiento, sin embargo, es posible utilizar otras dependiendo la aplicación. Ejemplos de medidas de similitud son: Manhattan, similitud de coseno, etc.

$$\text{grupo} = \arg \min \sum_{j=1}^k \sum_{i=1}^n \|x_i^j - c_j\| \quad (3.11)$$

Donde el termino: $\|x_i^j - c_j\|$ provee la distancia Euclidiana entre un punto de entrenamiento y los centroides del grupo.

Una consideración importante para k -Medias es la inicialización de los centroides. El algoritmo es sensible a la generación aleatoria de los centroides iniciales y que puede converger en un óptimo local. Por ejemplo, si dos centroides iniciales están muy cercanos, resultaría una agrupación pobre. Por lo que en la práctica se recomienda ejecutar el procedimiento varias veces. Otra forma de mitigar esta situación, es como se propone en el método k -Medias++[2], donde los centros iniciales se escogen mediante una probabilidad ponderada que favorece a seleccionar puntos lejanos a los centros ya seleccionados.

3.6. Apriori

Diariamente las empresas registran las compras de sus clientes, información que se le conoce como datos de la cesta de mercado. En los cuáles cada compra se representa como una transacción asociada a un identificador único y contiene una lista de elementos. La tabla 3.5[42] muestra un ejemplo con datos de este tipo, que indica las transacciones ficticias de compras realizadas en una tienda de comestibles. En dicha representación, por ejemplo, si una persona compra un o más productos del mismo tipo no es relevante, ya que solo se consideran elementos diferentes por transacción.

Número de transacción	Elementos
1	leche, pan
2	cerveza, huevos, pan, pañales
3	ceveza, leche, pañales, refresco
4	cerveza, leche, pan, pañales
5	leche, pan, pañales, refresco

Cuadro 3.5: Ejemplo de una lista de transacciones de compras hechas en una tienda de comestibles

Analizar datos de la cesta de compra es de interés para empresas, que buscan patrones que den soporte a la toma de decisiones. Por ejemplo, utilizar estos datos para descubrir los productos que comúnmente se compran juntos y utilizar este conocimiento para decidir que

productos se colocan en lugares adjuntos con los propósitos de ofrecer un mejor servicio y aumentar ventas.

El análisis de asociación es una tarea dentro de la minería de datos descriptiva, que permite encontrar relaciones ocultas en datos de la cesta de mercado. Estas relaciones pueden ser de dos formas: *reglas de asociación* y *conjuntos de elementos (items) frecuentes*. Las reglas determinan la relación de implicación entre dos elementos, por ejemplo, una regla inferida de la tabla 3.5, es que el 90 % de las personas que compran leche llevan pan; que se expresa de la forma $\text{Leche} \rightarrow \text{Pan}$. Por otro lado los conjuntos de elementos frecuentes son colecciones de elementos que de manera frecuente ocurren juntos. Por ejemplo, productos que comúnmente se compran juntos son: leche y pan.

El análisis de la cesta puede ser utilizada en otros dominios ajenos a compras. Como por ejemplo, encontrar los términos más frecuentes en las publicaciones en redes sociales es un indicador de las tendencias del momento. Para sugerencias de consultas en motores de búsquedas, donde es posible encontrar palabras que comúnmente se consultan juntas, como “aprendizaje maquina”, donde el buscador ante la presencia del término “aprendizaje” puede sugerir la palabra “maquina”.

De las dos tareas de análisis de asociación mencionadas se cubrirá la segunda, conjuntos de elementos frecuentes, la cuál se describe a continuación.

3.6.1. Descubrimiento de conjuntos frecuentes

Para describir la tarea de descubrimiento de conjuntos frecuentes primero se presenta la siguiente nomenclatura. Sea $I = \{e_1, e_2, e_3, \dots, e_n\}$ una colección de elementos y D una base de datos con transacciones de la cesta de compra, donde cada transacción T contiene una colección de elementos tal que $T \in I$. Asociado a cada transacción existe un identificador único llamado TID . Se dice que una transacción T contiene a X , un conjunto con algunos elementos de I , tal que $X \in T$.

Un conjunto frecuente es un conjunto X tal que su soporte sea mayor a un umbral dado. El soporte de X es el porcentaje de transacciones en D que lo contienen. Por ejemplo, en la tabla 3.5 el soporte para el conjunto de un elemento $X_1 = \{\text{leche}\}$ sería $4/5$ ya que 4 de las 5 transacciones lo contienen, lo que equivale a un 80 %. Donde si el umbral dado es 70 %, X_1 sería un conjunto frecuente, por otro lado considerando este mismo umbral, el conjunto de dos elementos $X_2 = \{\text{pan, leche}\}$ es $3/5$ que representa un 60 % no sería considerado frecuente.

La tarea de descubrir conjuntos frecuentes tiene por objetivo encontrar a partir de una base de datos con transacciones D y un umbral dado, una lista conjuntos de la forma $L = \{\{i_1\}, \{i_3\}, \{i_1, i_4\}, \{i_2, i_4, i_6\}\}$.

Uno de los enfoque para descubrir estos conjuntos consiste en generar conjuntos y descartar aquellos que no son frecuentes. Una forma sencilla de utilizar este enfoque es mediante la fuerza bruta, es decir, generar tantos conjuntos como combinaciones de los elementos en I existan, denominados conjuntos candidatos. Posteriormente recorrer la base de datos para obtener el soporte de cada conjunto candidato y al final descartar aquellos que no son frecuentes. Por ejemplo, suponga que se tiene un conjunto de cuatro elementos $I = \{e_0, e_1, e_2, e_3\}$

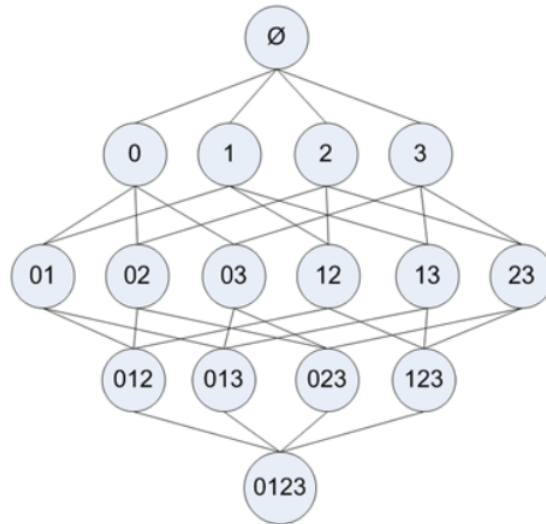


Figura 3.4: Todos los posibles conjuntos de los elementos: $\{0,1,2,3\}$.

resultarían 15 combinaciones, como se muestra en la figura 3.4. El símbolo ϕ indica el conjunto vacío, que significa que no se tienen ningún elemento. El total de conjuntos generados por fuerza bruta esta dado por 2^{n-1} donde n es el número de elementos distintos. Número que aumenta de forma exponencial mientras n incrementa, por lo que obtener el soporte para tal cantidad de conjuntos impactaría en los tiempos de respuesta así como memoria.

Una alternativa que permite controlar el crecimiento exponencial es el método Apriori[1]. Que consiste en generar conjuntos candidatos a partir de aquellos frecuentes. Por ejemplo en la figura 3.5 el conjunto $\{2,3\}$ no es frecuente, por lo que se sabe de antemano que los conjuntos: $\{0,2,3\}$, $\{1,2,3\}$ y $\{0,1,2,3\}$ no serán frecuentes. Una vez que se ha calculado el soporte para $\{2,3\}$ y se descubre no es frecuente, Apriori lo descarta para generar nuevos candidatos, esto es, no lo combina con otros conjuntos; lo que reduce la cantidad de conjuntos. La descripción del algoritmo Apriori se muestra a continuación.

3.6.2. Algoritmo Apriori

Para encontrar conjuntos frecuentes Apriori recorre la base de datos varias veces. La primera vez, calcula el soporte de los conjuntos de 1 elemento y se determina cuales son frecuentes. En recorridos subsecuentes, referidos como k , se utilizan los conjuntos frecuentes L_{k-1} de anteriores como semillas para generar conjuntos candidatos C_k . En el recorrido k , actual, se obtiene el soporte de los candidatos para conocer cuales son frecuentes. Aquellos que no son frecuentes se descartan y el resto se inserta en la lista de conjuntos frecuentes L_k , para ser utilizados en recorridos posteriores como semillas. El proceso continua hasta que ningún conjunto sea frecuente y se devuelve la lista L . El detalle Apriori se muestra en el algoritmo 5.

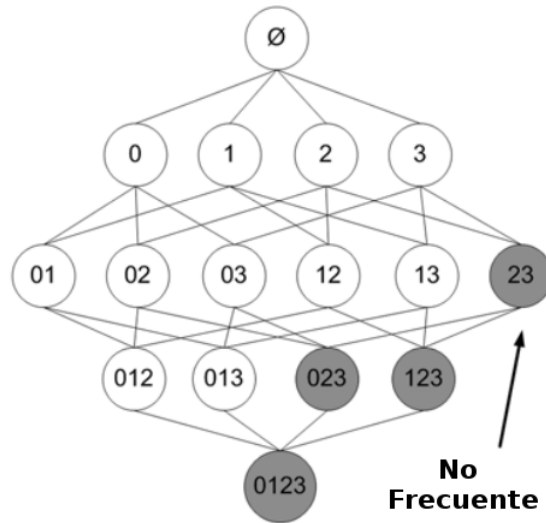


Figura 3.5: Todos los posibles conjuntos de elementos, los de color gris son los conjuntos no frecuentes.

Algoritmo 5: Apriori

Result: L_k

$L_1 = \{\text{conjuntos candidatos de tamaño } 1\}$

for $k \leftarrow 2$ **to** $L_{k-1} \neq 0$ **do**

$C_k = \text{apriori-gen}(L_{k-1});$ //Generar nuevos candidatos

forall the *transacciones* $t \in D$ **do**

$C_t = \text{subconjunto}(C_k, t);$ //Candidatos contenidos en t

forall the *candidatos* $c \in C_t$ **do**

$c.\text{contador}++;$

end forall

end forall

$L_k = \{c \in C_k \mid c.\text{contador} \geq \text{soporte minimo}\}$

end for

Son dos los aspectos importantes dentro del procedimiento, la generación de candidatos y el cálculo del soporte. Para el primero, la función *apriori-gen* se encarga de generar los conjuntos candidatos en un recorrido k . Recibe como argumento la lista de conjuntos frecuentes del recorrido previo L_{k-1} y devuelve una lista de conjuntos candidatos C_k . La función consta de dos operaciones:

1. Unión: Genera conjuntos C_{k+1} , al tomar la unión de los conjuntos de elementos frecuentes de tamaño k , P_k y Q_k que tienen los primeros $k-1$ elementos en común, donde;

$$C_{k+1} = P_k \cup Q_k = \{\text{elemento}_1, \dots, \text{elemento}_{k-1}, \text{elemento}_k, \text{elemento}_{k'}\}$$

$$P_k = \{\text{elemento}_1, \text{elemento}_2, \dots, \text{elemento}_k\}$$

$$Q_k = \{elemento1, elemento2, \dots, elemento_k\}$$

donde, $elemento1 < elemento2 < \dots < elemento_k < elemento_{k+1}$

2. Poda: Eliminar todos los conjuntos candidatos $c \in C_{k+1}$ cuyos subconjuntos de tamaño k no estén en L_{k-1} .

Para ilustrar la generación de candidatos en el recorrido $k=4$. Sea $L_3 = \{\{1,2,3\}, \{1,2,4\}, \{1,3,4\}, \{1,3,5\}, \{2,3,4\}\}$. Después del paso de unión, C_4 será $\{\{1,2,3,4\}, \{1,3,4,5\}\}$. El paso de poda descartará el conjunto $\{1,3,4,5\}$ por que el subconjunto $\{1,4,5\}$ no está en L_3 . Por lo que al final C_4 contendrá al conjunto candidato $\{1,2,3,4\}$. Este enfoque reduce de manera efectiva la cantidad de conjuntos considerados para contar su soporte.

El segundo aspecto a considerar es el conteo de soporte, que consiste en determinar la frecuencia de ocurrencia de cada conjunto candidato generado por la función *apriori-gen*. Cada miembro del conjunto C_k tiene dos campos: el conjunto y un contador de soporte. Para obtener el soporte de cada conjunto, se recorre la base de datos y en cada transacción se buscan los conjuntos candidatos. Si un subconjunto ocurre en una transacción i se incrementa su respectivo contador. Al finalizar el recorrido se calcula el soporte y mantiene aquellos conjuntos cuyo soporte supere al umbral dado. La base de datos se recorre hasta que no se generen candidatos.

3.7. Resumen

En este capítulo se revisaron diversos métodos de aprendizaje maquinal, relacionados a las tareas de: clasificación, agrupación y conjuntos frecuentes. Para la primer tarea se describieron tres métodos: Árboles de decisión, Bayes Ingenuo, Vecinos más cercanos (k NN). En el caso de agrupación se abordó k -Medias. Mientras que en conjuntos frecuentes se revisó Apriori.

En el caso de los procedimientos de clasificación. El primero, Árboles de decisión, lleva a cabo la tarea en base a decisiones. Para lo que construye un modelo, como su nombre lo indica, en forma de árbol. El segundo procedimiento fue Bayes Ingenuo que hace uso del Teorema de Bayes para estimar la clase más probable a la que pertenece una instancia. En donde el modelo se conforma de las probabilidades condicionales. El último de este tipo fue vecinos más cercanos (k NN) el cuál consulta las k instancias más cercanas y obtiene la moda de dicho conjunto de resultante obtiene la clase moda. Una característica de k NN es que no se genera un modelo.

Cada procedimiento ofrece ventajas y desventajas para propósitos de clasificación. Por ejemplo, el modelo generado por Árboles de decisión es fácil de interpretar para las personas. Sin embargo, como se mencionó anteriormente, en casos donde los datos no se encuentren linealmente separables k NN podría resultar una alternativa. Sin embargo cuando la cantidad de datos es grande k NN puede tomar más tiempo en presentar resultados. Donde en un punto medio se encuentra Bayes Ingenuo.

Para propósitos de agrupación se presentó *k*-Medias el cuál construye grupos utilizando un representante, donde las instancias son asignadas al grupo cuyo representante sea más similar o cercano. Posteriormente se actualizan los centros, calculando la media con base en los miembros de cada grupo. El proceso repite hasta que no haya cambios en los centros.

Por el lado de conjuntos frecuentes, se revisó Apriori que permite obtener conjuntos que comúnmente ocurren juntos. Método que se basa en la generación de candidatos. Donde su principal característica es el criterio utilizado para generar los conjuntos de candidatos. El cuál consiste en únicamente generar conjuntos a partir de otros que ya se sabe de antemano son frecuentes. Lo que ayuda a reducir la cantidad de conjuntos a buscar.

Capítulo 4

Big Data

Actualmente la minería de datos se hace sobre *Big Data*. El término es utilizado para referirse a una enorme cantidad de datos. Se estima que en los últimos años las bases de datos han crecido de forma exponencial, como resultado de este crecimiento en 2010 se rompió la barrera de los zettabytes a nivel mundial. Dicho crecimiento parece que no va a detenerse, por lo que existe una tendencia de realizar tareas de minería de datos sobre cantidades masivas de datos. Un ejemplo de esto, se da en el acelerador y colisionador de partículas (*Large Hadron Collider*, HLC) que produce anualmente 30 Petabytes de datos, lo cuales deben ser analizados por científicos para generar modelos que les permitan identificar y predecir colisiones interesantes.

De forma más precisa diversas fuentes describen a *Big Data* como bases de datos cuyo tamaño sobrepasa las capacidades de análisis de las herramientas convencionales en tiempos razonables [6, 18, 19]. A diferencia de las bases de datos tradicionales, incluye datos no estructurados como: publicaciones de redes sociales, documentos de texto, bitácoras de sistemas web, registros de sensores en dispositivos, etc. En base a estas condiciones *Big Data* se representa mediante el modelo de las 4V's [16]:

- **Volumen:** se refiere a cantidades masivas de datos que continúan creciendo a tal grado que su procesamiento se vuelve un problema para las herramientas comunes.
- **Variedad:** son los tipos de datos que se desean operar. Se consideran varios tipos: estructurados y no estructurados (por ejemplo: texto, vídeo, páginas web, etc).
- **Velocidad:** es el tiempo de análisis y recolección. En algunos casos se manejan flujos (*streams*) de datos y se requiere de análisis en tiempo real.
- **Valor:** es la información que se puede extraer de los datos, que tenga significado y aporte algún beneficio, ya sea económico o de otro tipo.

Realizar minería de datos sobre *Big Data*, implica generar modelos a partir de grandes cantidades de datos, lo que representa varios problemas. Uno de ellos es el consumo de los recursos. Como se mencionó anteriormente los algoritmos de aprendizaje realizan las operaciones correspondientes sobre memoria, por lo que si la cantidad es lo suficientemente grande se podría consumir la memoria disponible. Otro problema es la generación de resultados en

tiempos razonables. Para solucionar ambos problemas se requiere modificar los algoritmos para llevar a cabo las operaciones correspondientes en un tiempo aceptable.

Llevar a cabo minería de datos sobre *Big Data* es cada vez más necesario. En el sector comercial, capturar mayor cantidad de datos provee a las compañías mayor introspectiva de las preferencias de los clientes. Datos que pueden ser utilizados para crear productos y servicios personalizados. Por ejemplo *Amazon*, que registra las consultas, compras realizadas y toda la información adicional posible. A estos datos se les aplica un algoritmo de análisis de la cesta para realizar sugerencias de productos basados en aquellos que se compraron juntos.

4.1. Plataformas de almacenamiento de Big Data

El análisis de *Big Data* requiere herramientas que permitan acceder a los datos de manera eficiente. Una forma sencilla de almacenar grandes cantidades de datos es como archivos de texto en disco duro. En caso de consumir todo el espacio se puede optar por conseguir una unidad adicional externa a un bajo costo. El precio de un disco duro de un TB actualmente ronda entre los \$800 pesos. Sin embargo, realizar una tarea de análisis sobre estos archivos puede no ser una tarea sencilla, debido a que se requieren idear los mecanismos de acceso a los datos para posteriormente realizar las operaciones correspondientes al análisis.

Los sistemas de archivos distribuidos han sido adoptados para almacenar y acceder de forma eficiente a los datos de *Big Data*. Un sistema de archivos distribuido es un conjunto de computadoras conectadas entre sí que ofrecen espacio de almacenamiento, comportándose como si se tratará de una sola unidad. Además tienen mecanismos para manejar de forma transparente los servicios de acceso y almacenamiento. Para realizar la gestión de los datos consideran los factores:

- **Consistencia:** un sistema de almacenamiento distribuido requiere realizar copias de datos en diferentes computadoras por si llega a fallar alguna y no perder los datos.
- **Disponibilidad:** si alguna de las computadoras falla el sistema debe conocer la ubicación de las copias realizadas para hacerlas disponibles.
- **Tolerancia a particiones:** debido a que el sistema funciona sobre computadoras conectadas por red, se espera que el sistema tenga tolerancia por si se pierde la conexión a una de estas.

El sistema de archivos de Hadoop (HDFS)[13] es un ejemplo de un sistema distribuido de archivos. El cuál opera sobre un *cluster* de computadoras. Un *cluster* es un conjunto de computadoras (también llamados nodos) conectadas a través de una red que se comporta como si fuese una única computadora. HDFS es un sistema escalable que funciona sobre computadoras personales y puede ser utilizado por aplicaciones de procesamiento intenso de datos.

Otra herramienta utilizada para un acceso eficiente a los datos de *Big Data* son las bases de datos NoSQL, donde en ocasiones las bases de datos tradicionales en ocasiones no cumplen

con sus requerimientos. Por otro lado las base de datos NoSQL son más flexibles en cuando al modelo, ofrecen interfaces de programación (API) más sencillas así como soporte para grandes cantidades de datos. Algunos ejemplos son: MongoDB, Casandra, etc.

4.2. Plataformas de programación

Una de las tendencias para el procesamiento de *Big Data* es el computo distribuido paralelo. Por ejemplo, en problemas donde se requiere aplicar la misma función a los datos, si solo se considera sola computadora, un opción es cargar a memoria principal una porción de los datos y almacenar los resultados en disco duro, este enfoque puede impactar en los tiempos de procesamiento. Por otro lado, otra forma consiste en utilizar varias computadoras para realizar dicho procesamiento de forma paralela. El computo distribuido paralelo permite distribuir operaciones sobre varias computadoras para ser ejecutadas en paralelo, lo que ofrece mejoras en los tiempos de respuesta.

Usualmente el cómputo distribuido se realiza sobre un *cluster* de computadoras. Para ello, se necesita de la programación paralela. Por medio de esta, se puede dividir el problema en subtareas y asignarlas a los procesadores de diferentes nodos para que sean ejecutadas en paralelo. Existen diversos modelos de programación paralela como: programación de paso de mensajes, programación por memoria compartida, etc. Algunas especificaciones de estos modelos son: *MPI*[12] que utiliza el modelo de paso de mensajes y *OpenMP*[7] que soporta memoria compartida.

Para los modelos tradicionales de programación paralela obtener los datos se vuelve un punto crítico. Por ejemplo, si un nodo dentro del *cluster* tiene que operar sobre datos que no se encuentran dentro de su almacenamiento, secundario o primario, se tiene que idear un mecanismo para recibirlo por medio de la red. El tiempo de espera por esos datos, puede estar limitado por el ancho de banda de la red u otros factores. Lo que implica que el acceso a los datos puede convertirse en un cuello de botella. Además es necesario que el programador implemente este mecanismo de acceso y asignación de datos, lo que puede llevar a generar código que puede oscurecer la solución del problema que se plantea resolver.

Han surgido modelos de programación paralela específicamente para el procesamiento de grandes volúmenes de datos. A diferencia de los modelos tradicionales consideran la ubicación de los datos, las tareas se asignan automáticamente, permiten re-ejecución de tareas en caso de fallo, etc. Por lo que estos modelos son pieza clave para el procesamiento masivo de datos. Entre estos modelos se encuentran: MapReduce[8], Spark[50], Pregel[29], Dryad[17], etc.

4.2.1. MapReduce

MapReduce es un modelo de programación y marco de trabajo asociado para el procesamiento paralelo y distribuido de datos. El modelo ofrece dos funciones: *mapeo* y *reducción*, cuyo comportamiento debe ser definido por el programador. Por otro lado, el marco de trabajo MapReduce se encarga de la paralelización automática de tareas, tolerancia a fallas y escalabilidad, para realizar el procesamiento paralelo y distribuido sobre un *cluster*.

El modelo MapReduce realiza operaciones sobre los datos utilizando un enfoque restrictivo. Ya que únicamente ofrece las dos funciones mencionadas anteriormente, donde para resolver un problema la solución se debe expresar en términos de estas.

El marco de trabajo de MapReduce utiliza el modelo para realizar el procesamiento de los datos en dos etapas. En la primera etapa se aplica la función de mapeo y en la segunda la función de reducción. Adicionalmente el marco de trabajo permite distribuir las tareas sobre los nodos de un *cluster* para procesar de forma eficiente grandes conjuntos de datos.

Modelo de programación

Todos los datos procesados por MapReduce tienen la forma $(llave, valor)$. El usuario expresa los cálculos en términos de las funciones: mapeo y reducción.

La función de *mapeo*, cuyo comportamiento es definido por el usuario. Recibe como entrada la pareja $(llave, valor)$ y produce una lista de resultados intermedios del mismo formato. Posteriormente se agrupan los resultados intermedios y los valores asociados a cada *llave* son enviados a funciones de reducción.

$$mapeo :: (llave_1, valor_1) \rightarrow lista(llave_2, valor_2)$$

La función de *reducción*, también definida por el usuario. Recibe resultados intermedios con una llave I y una lista de valores asociados a esa llave. Generalmente en la función se itera la lista para realizar alguna operación de agregación como suma o promedio para devolver un valor asociado a esa llave.

$$reduccion :: (llave_2, lista(valor_2)) \rightarrow lista(valor_2)$$

Ejecución de MapReduce

La ejecución de un trabajo se lleva a cabo en dos etapas (trabajo se refiere al procesamiento completo de un conjunto de datos). En la primera se ejecutan operaciones de mapeo. Inicialmente se divide el conjunto inicial en M bloques. A cada nodo dentro del cluster se le asignan tareas de mapeo y un bloque de datos, para ser ejecutadas en paralelo. El resultado de esta etapa genera resultados intermedios, que serán ordenados y agrupados. Los resultados generados en una etapa de reducción corresponden a los resultados finales de una aplicación.

En la segunda etapa se ejecutan tareas de tipo reducción, donde la cantidad R de tareas de este tipo se define por el usuario. En esta etapa se leen los resultados intermedios, generados por las tareas de reducción, y son asignados a los nodos para llevar a cabo las R tareas de reducción especificadas. Un ejemplo de reducción, puede ser sumar una lista de valores asociadas a la misma llave. Generalmente el resultado de esta fase corresponde a los resultados finales.

En la figura 4.1 se muestra la ilustración de la operación del modelo.

Dentro de las etapas, la asignación de tareas la realiza el marco de trabajo de MapReduce. El marco de trabajo permite encapsular lo referente a la paralelización liberando al progra-

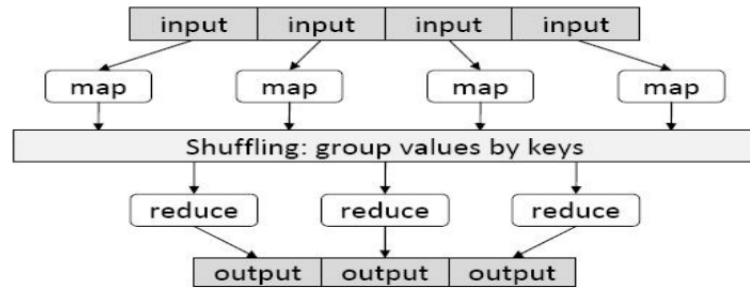


Figura 4.1: Ilustración del modelo MapReduce. La función de mapeo es aplicada al conjunto original, lo que produce resultados intermedios. Que serán agrupados para ser procesados por las funciones de reducción

mador de tener que implementar los mecanismos correspondientes. Las principales funciones de las que se encarga son:

- Distribución de tareas: Asigna tareas a los nodos de forma automática, y un bloque de datos a procesar. Conforme un nodo termina de procesar un bloque, asigna otro hasta completar el conjunto completo.
- Tolerancia a fallas: Se encarga de vigilar constantemente los nodos. En caso de perder la comunicación con alguno, vuelve a ejecutar la función en otro nodo.
- Escalabilidad: Ante la presencia de nuevos nodos asigna tareas correspondientes de forma automática sin necesidad de modificar el código para incorporarlos al procesamiento.

Para ilustrar la operación del marco de trabajo de MapReduce considere la representación esquemática en la figura 4.2 [46]. Donde cada computadora tiene dos procesadores y puede manejar dos tareas de mapeo o reducción simultáneamente. Si la máquina 0 fallará en la fase de mapeo, el marco de MapReduce estaría al tanto de esto y volvería a lanzar la tarea correspondiente a cualquiera de los dos nodos, dando continuidad al trabajo.

Ejemplo: Calcular promedio de temperatura

Suponiendo que se desea conocer la máxima temperatura registrada en China en los últimos 100 años por cada provincia. Se tiene un conjunto de archivos con temperaturas registradas de la forma: `<provincia> <fecha> <temperatura>`.

En un enfoque MapReduce, el conjunto se divide en M bloques, cada bloque es asignado a un nodo para ejecutar una función de mapeo. El resultado de esta función es la pareja (“<provincia>”, <temperatura>), donde se emiten exclusivamente las temperaturas cuya diferencia entre la fecha registrada y la actual no supere los 100 años. Posteriormente se agrupan los resultados intermedios por llave, en este ejemplo, por provincia. Las listas de temperatura por provincia son de la forma: (“<provincia>”, lista<temp₁, temp₂, ..., temp_n>).

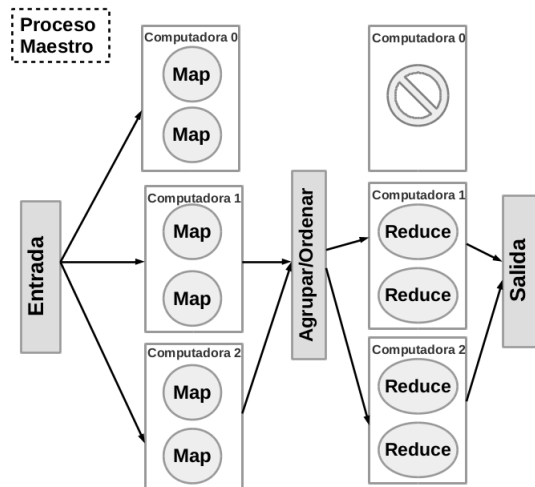


Figura 4.2: Representación esquemática de MapReduce. En el ejemplo se muestra un *cluster* con tres computadoras con dos procesadores cada una.

En la etapa de reducción, se asigna a R funciones de reducción a diversos nodos. Cada función recibe como parámetro una lista de temperaturas asociadas a una provincia. En una función de tipo reducción se iteran los elementos para obtener la máxima temperatura por provincia. El resultado por cada función de reducción es ("*<provincia>*", *<temperatura>*).

Hadoop

Apache Hadoop[13][44] es una librería de código abierto que provee un marco de trabajo para realizar el procesamiento distribuido de grandes cantidades de datos sobre un *cluster*, utilizando modelos sencillos de programación. Fue diseñada para escalar una aplicación desde una computadora hasta miles. También es capaz de detectar y manejar fallas a nivel de aplicación. Los principales módulos que componen Hadoop son:

- *Hadoop Distributed File System (HDFS)*: un sistema distribuido de archivos que provee un acceso de alto rendimiento a los datos.
- *Hadoop YARN*: manejador de recursos dentro del *cluster*, que adicionalmente se encarga de la calendarización de tareas.
- *Hadoop MapReduce*: implementación del modelo de programación MapReduce para ser ejecutado sobre en YARN.

Los tres módulos operan en conjunto para realizar el procesamiento de grandes cantidades de datos. Comúnmente Hadoop se describe como la implementación del Modelo MapReduce, ya que adopta todas funcionalidades del modelo y del marco de trabajo para el procesamiento

de los datos sobre un *cluster*. Sin embargo, también ofrece un sistema de archivos distribuido y un manejador de recursos del *cluster*.

Hadoop ha ido evolucionando desde las primeras versiones, en las cuales la plataforma únicamente ejecutaba trabajos MapReduce, posteriormente se realizaron cambios en su arquitectura lo que dio origen a YARN. En YARN se desacopla el marco de MapReduce en dos módulos: el modelo MapReduce y el manejador de recursos YARN. Lo que resulta en un manejador más genérico.

En YARN es posible ejecutar tareas de tipo MapReduce o de Grafos Dirigidos Acíclico (por sus siglas en inglés DAG). En MapReduce las tareas se ejecutan una después de la otra, la dependencia es lineal y únicamente se tiene una tarea padre. Mientras que en DAG se refiere a que se tiene una jerarquía donde una tarea puede depender de varias tareas predecesoras. Los principales componentes de YARN se muestran en la imagen 4.3[44] y se describen a continuación:

Contenedor: Es un conjunto de recursos de sistemas asignado para ejecutar una tarea. Los recursos que se pueden solicitar son: CPU y memoria principal. La relación entre un nodo y un contenedor es: un nodo puede tener varios contenedores, pero un contenedor no puede abarcar varios nodos. Por lo que un contenedor puede ser visto como una unidad de trabajo. Un contenedor compromete recursos en un nodo, así que implícito en un contenedor se encuentra el nodo en el que se registró. Los contenedores se solicitan a un nodo en específico. Dentro de una aplicación es el permiso otorgado para acceder a los recursos de CPU y procesamiento de un nodo específico. Un trabajo ya sea de tipo MapReduce o DAG se ejecutará sobre uno o varios contenedores. Estos son asignados por el marco de YARN a través del componente *Manejador de Nodos*.

Manejador de Nodos: Cada nodo dentro del *cluster* ejecuta un manejador de nodos. Es un proceso esclavo que atiende peticiones del *Manejador de Recursos* para asignar contenedores a aplicaciones. En conjunto con este último son responsables de asignar los recursos dentro del *cluster* de Hadoop. Mientras que el *Manejador de recursos* es un componente global, el de nodos es local y se encarga de inspeccionar así reportar el estado de cada contenedor en el nodo al *Manejador de Recursos*. Cuando se agrega un nodo, este se registra e indica sus recursos disponibles, que podrán ser utilizados para asignar contenedores. Durante el uso de este nodo, dicha información se actualiza constantemente conforme los manejadores operan en conjunto para gestionar de forma eficiente los recursos del *cluster*.

Manejador de Recursos: Se trata de un calenzarizador que se encarga de otorgar recursos a aplicaciones, para realizar esto de forma eficiente tiene un panorama global, permitiendo ejecución coordinada de procesos dentro *cluster*.

Aplicación Maestra: Es una instancia de un marco de trabajo específico. La aplicación maestra solicita recursos al Manejador de Recursos para obtener contenedores y ejecutar tareas. Por ejemplo, una instancia de una aplicación de MapReduce, para llevar a cabo

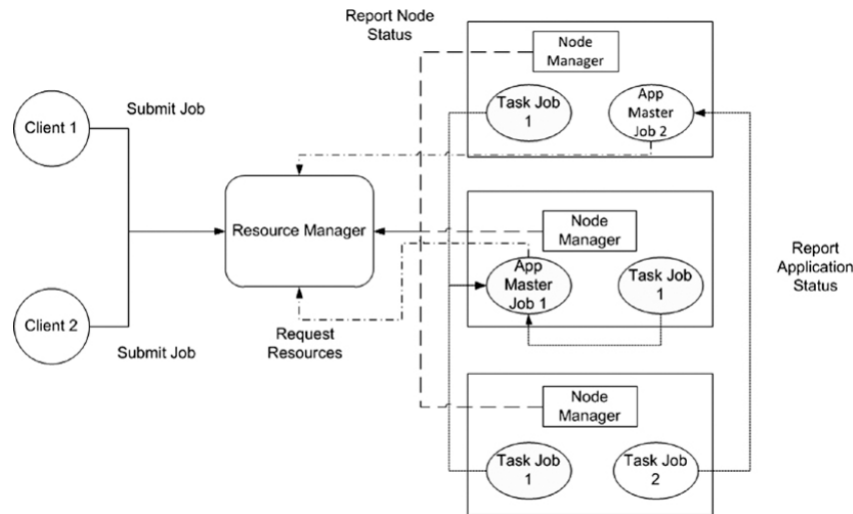


Figura 4.3: Arquitectura del Manejador de Recursos YARN. Que se encarga de ejecutar trabajos MapReduce.

el procesamiento distribuido de los datos sobre el *cluster*, de la forma como se explicó anteriormente.

En base a estos componentes, la ejecución de un trabajo en YARN es:

1. Un programa cliente lanza una aplicación para ser ejecutada dentro del cluster. También se especifica el tipo de aplicación, por ejemplo, de MapReduce.
2. El Manejador de Recursos solicita un contenedor dentro de un nodo para ejecutar la aplicación maestra.
3. La Aplicación Maestra se registra con el Manejador de Recursos. Este registro permite al programa cliente solicitar información al Manejador de Recursos sobre el estado de la Aplicación Maestra.
4. La Aplicación Maestra solicita al Manejador Recursos para ejecutar tareas.
5. El Manejador de Recursos negocia con los manejadores de nodos para obtener los contenedores solicitados por la Aplicación Maestra. El Manejador de Nodos se encarga de asignar los contenedores.
6. Las tareas que se encuentran ejecutándose en los contenedores reportan su estatus a la Aplicación Maestra utilizando un protocolo específico.

YARN busca expandir el uso de Hadoop más allá de MapReduce. Este último tiene diversas limitaciones, ofrece ventajas para aplicaciones altamente paralelizables, sin embargo para aplicaciones iterativas que reutilizan datos entre etapas acceder a disco para obtenerlos puede resultar una operación costosa. Por lo que han surgido otros modelos que pretenden resolver estas limitaciones, uno de ellos es Spark, que a continuación se describe.

4.2.2. Spark

Spark es un marco de trabajo de computo distribuido que permite procesar grandes cantidades de datos sobre un cluster. Uno de los marcos más utilizados para el procesamiento de datos sobre un cluster es MapReduce, en donde el procesamiento se realiza por etapas. Para compartir datos entre estas etapas se requiere almacenar los resultados intermedios en disco duro. Spark por otro lado permite mantener estos resultados en memoria para ser utilizados nuevamente. Esta característica es conveniente en aplicaciones de minería de datos, por ejemplo el algoritmo k -Medias donde se realizan cálculos con los resultados de iteraciones previas.

Para el procesamiento de los datos, Spark hace uso de los *Conjuntos de datos distribuidos resilientes* (por sus siglas en inglés RDD's). Un RDD es una colección de datos distribuida que permite mantener en memoria los resultados de operaciones realizados sobre este. Cada nodo dentro del *cluster* mantiene en memoria un bloque de la colección. Donde el marco de trabajo indica a los nodos del cluster las operaciones a realizar sobre un RDD.

Conjuntos de datos distribuido resilientes (RDD's)

Los RDD's son una estructura de datos distribuida de solo lectura que opera sobre un cluster. Cada nodo proporciona un porcentaje de memoria principal para albergar la estructura y los datos correspondientes y cada nodo mantiene una partición de la estructura. De forma sencilla, estos se pueden visualizar como una colección de elementos similar a las estructuras de datos existentes en lenguajes de programación como: listas, arreglos, etc; con la condición que se encuentran distribuidos sobre los nodos de un *cluster* y pueden ser operados de forma paralela.

Son dos las formas en las que se pueden crear RDD's : *i*) desde archivos en almacenamiento y *ii*) como resultado de operaciones deterministas sobre estos. Por ejemplo, es posible crear un RDD de tres archivos almacenados en 3 nodos diferentes dentro del *cluster*, cada archivo representa una partición del RDD. Aplicar una transformación sobre un RDD resulta en otro nuevo.

Las operaciones que se pueden realizar sobre un RDD, son de dos tipos. El primer tipo se denomina *transformaciones*, permiten operar de forma paralela los RDD's. Las transformaciones reciben como parámetro una función que será aplicada a cada elemento de un RDD y el resultado será uno nuevo. Los elementos de un RDD son independientes, por lo que es posible procesarlos de forma concurrente. Las transformaciones pueden ser ejecutadas sobre cada partición de forma paralela, por cada nodo que las contiene. En el ejemplo anterior, cada nodo podría ejecutar las transformaciones de forma paralela en la partición que le corresponde del RDD.

El segundo tipo de operaciones que se pueden realizar sobre los RDD's son las *acciones*. Estas retornan un valor que resume los valores de los elementos de un RDD. Por ejemplo, la acción *count* devuelve la cantidad de elementos de un RDD dado.

La relación entre ambas es que las transformaciones no se llevan a cabo hasta que se invoca una acción sobre un RDD. Una característica de las transformaciones es que son

operaciones perezosas (*lazy*) ya que durante la ejecución de un programa no se realizan de forma inmediata sino hasta que son requeridas. En momento en que se invoca una acción se llevan a cabo las transformaciones, materializando los resultados producidos en memoria como una nuevo RDD para ser utilizados para los cálculos requeridos por una acción.

Transformación	Descripción
<code>map(f)</code>	Transforma un RDD aplicando la función f a cada elemento de la colección.
<code>filter(f)</code>	Se evalúa cada elemento de la colección y se genera un nuevo RDD con aquellos cuyo resultado sea verdadero.
<code>flatMap(f)</code>	Transforma un RDD separando los valores de cada elemento en una secuencia.
<code>sample(r,fr,s)</code>	Se genera un nuevo RDD con una muestra con remplazo o sin remplazo del RDD original.
<code>groupByKey()</code>	Se llama sobre un RDD cuyos elementos tienen el formato de pareja (K,V) y genera uno nuevo con (llave, seq(V)). Donde seq(V), representa una lista de valores asociados a la misma llave.
<code>reduceByKey(f)</code>	Reduce los valores correspondientes de cada llave en base a la función f
<code>union()</code>	Une dos RDDs en uno nuevo
<code>join()</code>	Realiza la operación join sobre dos RDD con el formato: (K,W) y (K,V) y produce uno nuevo de la forma (K,(W,V))
<code>cogroup()</code>	Realiza la operación join, pero el resultado tiene el formato: (K, Seq(W), Seq(V))
<code>crossProduct()</code>	Es el producto cartesiano de dos RDD's.
<code>sort(c)</code>	Ordena los elementos del RDD
<code>partitionBy(p)</code>	Permite dividir los elementos por particiones acorde a la función p . Similar a hacer la asignación en un <i>hashmap</i>

Cuadro 4.1: Transformaciones sobre los RDD's

Acción	Descripción
<code>count()</code>	Cuenta los elementos de un RDD.
<code>collect()</code>	Colecta todos los valores y se envían al programa principal.
<code>save(ruta)</code>	Guarda un RDD es sistema de almacenamiento secundario, ej., HDFS
<code>take(n)</code>	Retorna los primeros n elementos de un RDD.

Cuadro 4.2: Acciones sobre los RDD's

Tipo de transformaciones

En la tabla 4.1 se muestra un listado de transformaciones y en la 4.2 ejemplos de acciones. Algunas transformaciones requieren más de un RDD o partición para ser calculadas como:

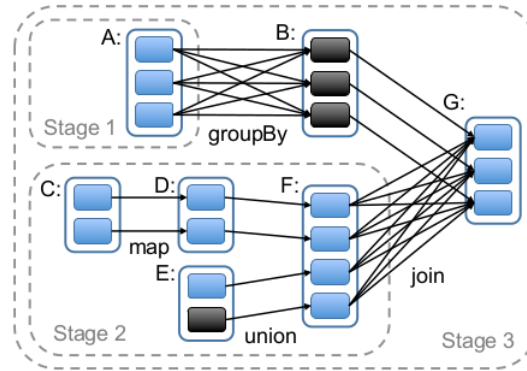


Figura 4.4: Ejemplo de como Spark calcula las etapas. Los rectángulos con líneas solidas son RDD's. Las particiones son los cuadros grises y negros. Se muestran 3 etapas generadas.

union, *join*. Estas se clasifican, de acuerdo a la cantidad de particiones requeridas para ejecutarlas, en dos tipos:

- **Estrechas** (*Narrow*): son transformaciones lineales, esto es, se requiere una solo partición padre para generar los resultados. Por ejemplo, la transformación *map* recibe una función que será aplicada a cada elemento de un RDD_{padre} y el resultado es otro RDD_{hijo} con la misma cantidad de elementos, donde cada valor corresponde al resultado de aplicar dicha función a los elementos del RDD_{padre} .
- **Amplias** (*Wide*): este tipo requiere de varias particiones padres para ser calculadas. Por ejemplo, la transformación *groupByKey* que coloca los elementos con la misma llave distribuidos sobre varias particiones de uno o varios RDD_{padres} en un RDD_{hijo} .

Ejecución de Spark

El procesamiento de los datos se realiza mediante etapas, las cuáles son definidas por Spark y están conformadas por transformaciones del mismo tipo. Por cuestiones de eficiencia se coloca dentro de una misma etapa tantas transformaciones *estrechas* como sea posible y tipo *amplias* en otra. En la figura 4.4[50], se muestra un ejemplo de etapas definidas. Cada etapa se ejecuta de forma secuencial en el orden en que sus resultados son requeridos. Para las dependencias amplias, se requieren materializar los resultados de la etapa anterior en memoria de los nodos que contienen la partición.

En base a las etapas Spark define las tareas requeridas para el procesamiento de los datos. Una tarea consiste en las transformaciones a realizar sobre un RDD. El marco de trabajo Spark asigna estas tareas de forma automática a los nodos del cluster, para ser ejecutadas de forma paralela sobre la partición del RDD correspondiente. Por ejemplo, asignar la tarea compuesta por las transformaciones *map* y *filter* sobre un RDD con tres particiones, donde cada nodo mantiene la partición ejecuta la tarea.

Las tareas se ejecutan en el orden que fueron definidas y el orden en que se ejecutan se puede expresar como un grafo acíclico dirigido (por sus siglas en inglés DAG). Un grafo acíclico, es aquel en el que para cada vértice v no hay un camino directo que empiece y termine en v . En Spark cada vértice denota una tarea. Dicha representación, es utilizada para identificar dependencias y en base a estas calendarizar el orden en que son ejecutadas.

El DAG representa el linaje de un RDD y es utilizado como mecanismo de fallas. El linaje de un RDD es la bitácora de las transformaciones requeridas para materializarlo. Si el falla el nodo que mantiene una partición P_1 de un RDD, Spark accede a esta bitácora para determinar que transformaciones lo originaron y vuelve a ejecutar las transformaciones correspondiente en ese u otro nodo.

Apache Spark

Existe una implementación de código abierto de Spark llamada Apache Spark[41]. Esta ofrece los RDD mediante una API para su manipulación en los lenguajes: Scala, Java y Python. Adicionalmente también es un marco de trabajo que permite operar los RDD's sobre un cluster.

La definición de operaciones sobre los RDD's se lleva a cabo en un programa principal. En este se especifican transformaciones e invocan acciones, las cuales serán identificadas por Spark durante su ejecución, con el propósito de generar el DAG con la traza de las transformaciones correspondientes y las etapas.

El programa principal se conecta a un manejador de recursos de un *cluster* para obtener trabajadores (*executors*). Los trabajadores son procesos que pueden almacenar en la memoria de un nodo una partición de un RDD y realizar operaciones. Los manejadores de recursos sobre los que puede operar Spark son: Mesos, YARN y modo autónomo. YARN es el manejador de recursos de Hadoop 2.x, por lo que es posible utilizarlo para ejecutar trabajos de tipo Spark.

A partir del programa principal, Spark define las tareas que le serán asignadas a los trabajadores. Cada trabajador realiza las transformaciones correspondientes y materializa los resultados intermedios en RDD's para ser utilizados por otras transformaciones o acciones. Los valores generados por una acción son enviados al programa principal, que se encarga de resumirlos para obtener uno único. Por ejemplo, para la acción *count* primero se crea el RDD en memoria, posterior cada trabajador cuenta la cantidad de elementos de su partición y envía ese número al programa principal, este último contabiliza todos los valores que le son enviados por los trabajadores para obtener el total. En la figura 4.5[41] se ilustra la ejecución de Spark.

De forma más detallada los pasos para ejecutar una aplicación en Spark son:

1. El programa principal (*driver*) ejecuta la aplicación y crea el contexto de Spark.
 2. El contexto de Spark, se conecta el manejador del *cluster* (ej: Mesos/YARN) este asigna recursos.
-

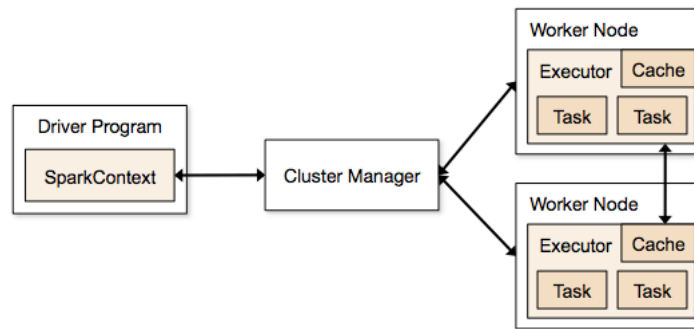


Figura 4.5: Ejecución de Spark en un *cluster*. El programa principal asigna tareas a los nodos y cada nodo devuelve sus al programa principal.

3. Spark adquiere trabajadores en los nodos de un *cluster*, que son procesos que ejecutan cálculos y almacenan datos de la aplicación.
4. El programa principal envía el código de la aplicación a los procesos trabajadores.
5. El Contexto Spark envía tareas para ser ejecutadas por trabajadores.

Scala

Scala[33][38] es un lenguaje de programación, basado en los paradigmas de orientación a objetos y programación funcional. El enfoque funcional permite escribir programas de forma similar a un lenguaje de *scripting*. Por otro lado, el enfoque orientado a objeto permite diseñar sistemas robustos y escalables.

Scala como lenguaje orientado a objetos es puro, cada valor es un objeto y cada operación es una invocación a un método. Se hace uso de clases y *traits* para definir un comportamiento. Los *traits* son como las interfaces de Java pero pueden tener implementaciones de métodos y variables. Los objetos son construidos como una composición *mixin*, donde se toman los miembros de una clase y también agrega los miembros de un número de *traits*. Esta característica permite encapsular el comportamiento de varias clases en diferentes *traits*.

Adicionalmente a ser un lenguaje orientado a objetos, Scala también es un lenguaje funcional. Esta se basa en dos ideas principales. La primera es que las funciones son valores de primera clase; es decir, tienen el mismo estatus que una variable de tipo Entero o String. Siendo posible: enviarlas como parámetro de otra, retornarlas como resultado o asignarlas a variables. Donde funciones que reciben y/o devuelven otra función se conocen como funciones de orden superior.

La segunda idea de la programación funcional es, que en las operaciones de un programa deben corresponder los valores de entrada a los de salida en vez de modificar los valores de entrada. Por ejemplo en JAVA, el tipo de dato String se representa como un arreglo de caracteres, realizar una operación de remplazo de un carácter por otro resulta en una nueva cadena con el nuevo valor. Considerando únicamente el manejo de Strings, JAVA adopta una idea de programación funcional. Las estructuras de datos inmutables son clave dentro de la

programación funcional. Scala ofrece diversas colecciones de este tipo como: listas, arreglos, mapas y sets.

Los lenguajes funcionales promueven las estructuras de datos inmutables y métodos que reciben funciones como parámetros. Algunos lenguajes establecen esta condición, Scala permite escribir programas con un estilo imperativo o funcional. Sin embargo, se aconseja utilizar las ventajas que ofrece la programación funcional, especialmente en el procesamiento de los datos.

Aprendizaje Maquinal con MLlib

MLlib[32][20] es una librería con una variedad de implementaciones de algoritmos de aprendizaje maquinal. Esta librería se incluye dentro del ecosistema de aplicaciones de Apache Spark. Los algoritmos que se encuentran en MLlib corresponden a las tareas de: clasificación, regresión, agrupación, reglas de asociación, filtro colaborativo, etc. Adicionalmente ofrece funciones para adecuar los datos al formato requerido por los algoritmos, como por ejemplo, convertir el texto a una representación de vector de Frecuencia de Términos - Frecuencia Inversa de Documento (*Term frequency - Inverse Document Frequency*, TF-IDF). Así como tipos de datos especiales para dichas funciones, el tipo *LabeledPoint* que representa un vector y su clase asociada.

Fue diseñada para ejecutar los algoritmos de forma paralela utilizando RDD's. Algunos ejemplos de estos algoritmos son: Bosques Aleatorios, *k*-Medias, Fp-growth, etc. MLlib es una herramienta útil en escenario de grandes cantidades de datos. Si se tiene acceso a bases de datos más pequeñas, que puedan ser procesadas en tiempo razonable por una computadora, se podrían considerar herramientas como WEKA[45] o SciKit-Learn[39].

MLlib es una librería de bajo nivel ya que requiere de programación para su uso. A diferencia de WEKA, que cuenta con interfaz gráfica donde se interactúa con la aplicación para obtener los resultados, MLlib requiere desarrollar un programa donde se carguen los datos, realice procesamiento previo y se invoquen las funciones de los algoritmos. Por ejemplo, para realizar la tarea de clasificación de texto se requiere escribir un programa que realice los siguientes pasos:

1. Iniciar con RDD que represente los textos.
2. Ejecutar una función de MLlib que convierta los textos en un vector con valores numéricos, el resultado es otro RDD con vectores.
3. Llamar una función de clasificación enviando como parámetro el RDD con vectores.
4. Evaluar el modelo generado con datos de prueba con alguna función de evaluación.

4.3. Resumen

En este capítulo se abordó el tema de *Big Data* y se describieron algunas de las plataformas de programación sobre este. *Big Data* es el término utilizado para describir bases de datos

cuyo tamaño sobrepasa las capacidades de herramientas convencionales. Una forma común de describirlo es mediante el modelo de las cuatro V's: *Volumen*, *Variedad*, *Velocidad*, *Valor*. Dichas características se refieren a las cantidades de datos, diferentes formatos (estructurado, no estructurado), velocidad de generación y procesamiento así como información valiosa que se puede extraer. Por lo que bases de datos que cumplan con estas características se consideran dentro de una categoría de *Big Data*.

En la actualidad las tareas de minería se realizan sobre *Big Data*. Esto representa varios retos, dos de los principales son procesamiento y memoria. Una de las tendencias para resolverlos es el computo distribuido. Motivando el desarrollo de plataformas específicas para el procesamiento de los datos de *Big Data*, las cuáles operan sobre un *cluster*. De forma general estas plataformas utilizan un enfoque de divide y vencerás, ya que separan el conjunto completo de datos en bloques más pequeños que pueden ser manipulados por una sola computadora.

Una de las herramientas pioneras y ampliamente utilizadas es MapReduce que cuenta con una implementación llamada Hadoop. Otro ejemplo es Spark que también cuenta con una implementación denominada Apache Spark. En ambos casos, operan sobre un *cluster* de computadoras, ofrecen operaciones de alto nivel para llevar a cabo las operaciones, se encargan de la paralelización de tareas y tolerancia a fallas. Es tarea del usuario expresar el problema a resolver en términos de las operaciones correspondientes y es la plataforma quien se encarga de la distribución del trabajo.

Son varias las diferencias entre las dos plataformas mencionadas, sin embargo una de las más notorias, es que Spark hace uso de una colección de datos distribuida, llamadas Conjuntos de Datos Distribuidos Resilientes. Que permite mantener en memoria resultados intermedios entre operaciones. A diferencia de MapReduce que almacena sus resultados intermedios en disco duro. Donde pareciera que Spark puede ofrecer mejores tiempos de respuesta, con la condición de requerimientos altos de memoria.

Adicionalmente para propósitos de aprendizaje, Spark ofrece una librería con una gran variedad de implementaciones de algoritmos de aprendizaje llamada MLlib. Dichos procedimientos consisten en las versiones paralelas de las versiones tradicionales.

Implementación de algoritmos en Spark

En el presente capítulo se describen las implementaciones de algoritmos de aprendizaje maquinaal abordados en el capítulo 3 sobre Spark. Como se ha mencionado anteriormente, estos métodos fueron desarrollados con la premisa de la existencia de pocos datos, lo cuál en la actualidad no se cumple. Por lo que para realizar tareas de aprendizaje sobre bases de gran tamaño en tiempos aceptables requiere adaptarlos sobre plataformas adecuadas. En este caso se seleccionó Spark, motivado por las siguientes razones:

1. Ser capaz de realizar tareas de minería sobre grandes cantidades de datos: Llevar a cabo tareas de minería de datos sobre grandes volúmenes requiere nuevos métodos. La mayoría de los métodos tradicionales, se desarrollaron bajo la premisa de cantidades moderadas de datos, premisa que hoy en día no se cumple. Para poder hacer tareas de análisis de grandes conjuntos en tiempos razonables, la tendencia es adaptar dichos algoritmos sobre plataformas de computo sobre un *cluster*.
2. Conocer Spark para tareas de minería de datos: Spark permite el procesamiento de grandes cantidades de datos accediendo a los recursos que ofrece un cluster. Sin embargo, no fue diseñado específicamente para tareas de minería por lo que si esto último es requerido se deben expresar los algoritmos en términos de las operaciones que ofrece. Al implementar los algoritmos en Spark se busca obtener una perspectiva de que tan flexible es para realizar tareas de minería.

Spark ofrece características ventajosas para el desarrollo de algoritmos de aprendizaje maquinaal sobre grandes cantidades de datos. La primera es que los RDD's permiten mantener en memoria una gran cantidad de datos, los cuales pueden ser procesados de forma paralela. Considerando que los algoritmos de aprendizaje maquinaal realizan cálculos sobre el conjunto de entrenamiento para la generación del modelo. Hacer uso de los RDD's para la generación del modelo resulta eficiente debido que los datos se operan de forma paralela y distribuida, además en caso de que se requiera hacer cálculos iterativos, los RDD's mantienen los resultados en memoria facilitando el acceso a cálculos subsecuentes.

Otra característica es su facilidad de uso, como se mencionó anteriormente, el marco de trabajo se encarga de: paralelización automática de tareas, tolerancia a fallas y escalabilidad.

Por lo que para realizar tareas de aprendizaje sobre grandes volúmenes principalmente se requiere expresar los algoritmos en términos de transformaciones. Dicha plataforma de forma automática se encarga de la distribución del procesamiento sobre el *cluster*. Esto resulta conveniente ya que el usuario debe enfocar sus esfuerzos en expresar los algoritmos en el enfoque funcional de Spark de la forma que resulte más eficiente.

A continuación se muestra la implementación de cinco algoritmos de aprendizaje maquina en Spark. En el capítulo 3 se explicaron varios que comprenden dos de las principales funciones de la minería de datos. De la función de predicción se encuentran: Bosques aleatorios, Bayes Ingenuo, *k*NN mientras que desde la función de descripción: *k*-Medias, Apriori. Lo cuáles fueron seleccionados debido a que son métodos populares, adicionalmente son sencillos pero lo suficientemente robustos. Además han presentado buenos resultados en la práctica.

5.1. Bayes Ingenuo en Spark

Bayes Ingenuo es un método de clasificación que estima la clase más probable a la que pertenece una instancia. Donde los cálculos más intensivos se llevan a cabo al obtener dichas probabilidades, para lo que primero se requiere contar los histogramas de: clases y los valores de los atributos asociados a una respectiva clase.

Para iniciar a conocer el ambiente Spark se seleccionó la aplicación de clasificación de texto con Bayes Ingenuo. El cuál sencillo de implementar y paralelizable de manera natural. Por ejemplo, el cálculo de los histogramas se puede realizar de forma paralela así como la estimación de las probabilidades, donde éstas últimas se obtienen en base a los histogramas.

Este procedimiento se encuentra implementado dentro de la librería Mllib, para el caso de clasificación de texto requiere como insumo la matriz de frecuencia de términos, lo que implica desarrollar una tarea de procesamiento previo de los textos de entrenamiento. Recordando que la matriz se basa en vocabularios y que la cantidad de columnas corresponde a la de términos en este. Por ejemplo, utilizar el diccionario de la RAE que contiene alrededor de 93,000 palabras diferentes con un *corpus* de entrenamiento de 10,000 documentos, resultaría en una matriz de 93,000 columnas \times 10,000 filas. En este tipo de representación es común que exista una gran cantidad de palabras que ocurren en poco documentos, por lo que se tienen una gran cantidad de ceros que no aportan información, a este tipo de matrices se les conoce como ralas.

Sin embargo, considerando que el propósito de la matriz de frecuencia de términos es utilizada para, realizar los conteos requeridos para la estimación de las probabilidades. Se optó por implementar el método bajo otro enfoque que no requiere la tarea de procesamiento previo y realiza los conteos de forma directa de los textos, mitigando el consumo de memoria que pudiera requerir una matriz rala. El conteo de palabras se adapta de forma natural al principio de descomposición-agregación de MapReduce que se puede expresar mediante las transformaciones *flatMap* y *groupByKey* disponibles en Spark.

El proceso de clasificación consta de dos etapas:

1. Entrena: a partir de los documentos de entrenamiento se obtienen los histogramas
-

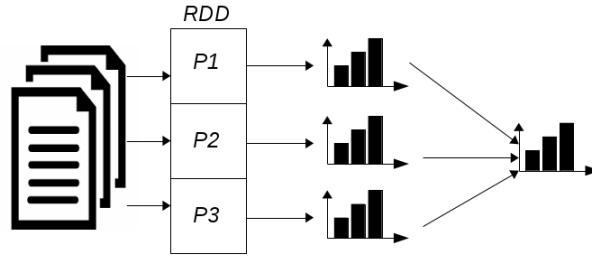


Figura 5.1: Representación del enfoque de cálculo de histogramas en Spark. El conjunto de entrenamiento se divide en particiones, por cada una se obtiene su histograma local y en base a estos el global.

requeridos para el cálculo de probabilidades:

- *vocabulario*: cantidad de palabras distintas dentro del *corpus*.
 - n_k : ocurrencia de la palabra i por clase.
 - n : cantidad de palabras diferentes por clase.
2. Clasifica: dado un nuevo documento que se desea clasificar, se descompone en palabras y en base a los histogramas calculados anteriormente se estima la probabilidad de cada palabra en base a la formula:

$$P(a_i|v_i) = \frac{n_k + 1}{n + |\text{vocabulario}|}$$

En base a esta estimación se utiliza el clasificador Bayes Ingenuo para determinar la clase más probable, como se mostró en la sección 3.3.2.

$$V_{NB} = \arg \max_{v_j \in V} P(v_j) \prod_i^n P(a_i|v_i)$$

En la etapa entrena el enfoque de Spark para calcular los histogramas consiste en descomponer los documentos en forma de la pareja: $((\text{palabra}, \text{clase}), \text{valor})$. Se inicia por crear un RDD donde cada elemento representa un texto del conjunto de entrenamiento. Tomando en cuenta que se esta procesando una gran cantidad de datos, es probable que un RDD contenga varias particiones, y las cuáles deban ser operadas en paralelo. Bajo éstas condiciones, los conteos se realizan de forma local a cada partición y posterior se obtienen los histogramas globales, como se muestra en la figura 5.1.

En la segunda etapa, de clasificación se procede a estimar la probabilidad que un documento pertenezca la clase. Para esto el clasificador requiere calcular dos tipos de probabilidades: $P(v_j)$ y $\prod_i^n P(a_i|v_i)$. La primera se obtiene a partir de la cantidad de documentos de la clase

Algoritmo 6: Etapa entrena: Procedimiento para obtener los histogramas en Spark

Data: RDD[(documento,clase)] documentos

Result: v, n_k, n

```

1 palClase ← documentos.flatMap{ doc → // Por cada documento.
2     palabras ← separar el documento en palabras.
3 /* Emitir la pareja, con la llave compuesta por: (pal,clase) y valor 1. */
4     palabras.map(pal→((pal,clase),1)
5 }
6 palabras ← palClase.map{
7     ((palabra,clase),valor)→(palabra,1)
8 }
9 v ← palabras.reduceByKey((x,y)→x+y).count
10 n_k ← palClase.reduceByKey((x,y)→x+y)
11 cantClase ← palClase.map{
12     ((palabra,clase),valor)→(clase,1)
13 }
14 n ← cantClase.reduceByKey((x,y)→x+y)

```

i entre el total. La segunda, es un poco más complicada y es uno de los cálculos más intensivos debido a la cantidad de operaciones requeridas. Solo para un documento dicha cantidad corresponde a la cantidad de términos en documento por el número de clases.

El enfoque mediante Spark para estimar dichas probabilidades condicionales, de las palabras dada la clase, se explica a continuación. Primero se muestra el listado de pasos de forma más general y en el algoritmo 10 las transformaciones y acciones utilizadas.

1. Descomponer en palabras el documento.
2. Por cada palabra estimar la probabilidad y mediante la transformación *flatMap* emitir la pareja (*clase, probabilidad*).
3. Reducir por la llave (clase) mediante la transformación *reduceByKey(-+-)*, la operación de reducción corresponde a la suma. Para obtener el término: $\prod_i P(a_i|v_i)$.
4. Estimar la probabilidad de la clase, en base a las probabilidades *a priori* y las probabilidades de todas las palabras (obtenidas en el paso anterior): $P(v_j) * \prod_i P(a_i|v_i)$.
5. Devolver la clase con la máxima probabilidad.

Es posible expresar el método de clasificación Bayes Ingenuo en Spark, el cuál se adapta de forma natural al enfoque de agregación-descomposición de este. Además no se requiere calcular la matriz de frecuencia de términos el insumo para el procedimiento es texto, evitando procesamiento adicional.

Algoritmo 7: Etapa clasifica: Estimar probabilidades en Spark

Data: $v, n_k, n, clases, documentoPrueba$
Result: clase

```

1 palabras ← descomponer en palabras el documentoPrueba
2 probPC ← palabras.flatMap{ palabra →
3     clases.foreach{ clase →
4         prob ←  $n_k+1/n+v$  // P(palabra|clase)
5         (clase,prob)
6     }
7 }
8 probPalClases ← probPC.reduceByKey((x,y)→x+y).collect
9 pDoc ← clases.map{ (clase, cantDoc) →
10     probClase ← cantDocClase / totalDocuemtnos
11     probDocumento ← probClase × probPalClases(clase)
12     (clase, probDocumento)
13 }
14 clase ← máxima probabilidad en pDoc
```

5.2. k NN Join en Spark

El método de k NN Join consiste en: dados los conjuntos R y S devuelve los vecinos más cercanos de los puntos R con respecto a S . Si $r \in R$, $knn(r, S)$ representa la función que devuelve los k vecinos más cercanos de r respecto a S . El procedimiento para obtener los vecinos más cercanos se explicó en la sección 3.4; donde se indica que r representa un punto en \mathbb{R}^n y los vecinos de r consisten en los k puntos más cercanos a este dentro de un conjunto dado. La métrica de distancia utilizada por defecto es la distancia Euclidiana.

De forma más precisa k NN join se define como:

$$kNNJoin(R, S) = \{r, knn(r, S) | \forall r \in R\}$$

Para propósitos de clasificación k NN Join funciona de la siguiente manera. S representa el conjunto de entrenamiento y R el de clasificación. Mediante k NN Join se obtienen los vecinos más cercanos de R respecto a S . Para realizar la clasificación, los vecinos de los puntos r realizan un voto con su clase donde se asigna aquella con mayores votos. El resultado es la clasificación para todos los puntos contenidos en R .

La implementación en Spark de knn Join se basó en el procedimiento de Bucle de Bloque Anidado Hadoop (H-BNLJ) [51], el cuál es una versión MapReduce del algoritmo Bucle de Bloque Anidado, utilizado en Bases de datos para realizar la operación *join* sobre dos tablas. De forma general H-BNLJ consiste en dividir en bloques los conjuntos R y S para colocar un bloque de cada conjunto en una cubeta. En MapReduce las cubetas son asignadas a diferentes nodos para calcular las distancias. De cada cubeta se obtienen los vecinos locales y con base a estos los vecinos globales requeridos para la clasificación.

Los pasos de H-BNLJ son:

1. Dividir los conjuntos R y S en n particiones, asignando $|R|/n$ (o $|S|/n$) registros a cada partición.
2. Copiar cada partición n veces.
3. Colocar una pareja de particiones en una cubeta, una de cada conjunto, S y R respectivamente. Cada pareja corresponde al producto cartesiano de las particiones de R y S , resultando en un total de n^2 cubetas.
4. Por cada cubeta obtener los k vecinos más cercanos locales del bloque R respecto al bloque S . Para lo que se requiere calcular la distancia de cada punto r a todos los puntos en S .
5. En base con los vecinos locales obtener los vecinos globales.
6. Realizar la clasificación de los puntos en R , donde los vecinos de cada punto emiten un voto por su clase.

La clave para expresar H-BNLJ en Spark radica en dar el formato de parejas adecuado a los elementos de un RDD y funciona de la siguiente manera. Se inicia por generar dos RDD's uno para el conjunto R y otro para el conjunto S . A cada elemento de esos RDD's se les asigna el formato `((cubeta, (id, instancia))`. Posteriormente se realiza el *join* de los dos conjuntos por la llave `cubeta`, esto genera un nuevo RDD donde cada elemento representa una cubeta conteniendo dos listas una de cada conjunto, con el formato `((cubeta, (bloqueS, bloqueR))`. Recordando las cubetas serán procesada por diversos nodos. Ahora bien por cada una de estas últimas, se calcula la distancia entre los bloques locales R y S , lo que genera los k vecinos locales, con la pareja `((cubeta, (idr, distancias))`. Posteriormente se cambia el formato y ahora la llave corresponde a `idr` para realizar una agrupación por llave produciendo el formato `((idr, lista(d1, d2, ..., dn))`. Dicha agrupación corresponde a una aproximación de k NN Join, debido a que se tiene una mayor cantidad de vecinos ya que se tienen todos los vecinos locales. En este último RDD cada elemento representa un punto ri con su lista de vecinos, donde se procede a obtener los k -vecinos y realizar la clasificación, al finalizar se obtiene el formato `((idr, clase))`. A continuación se muestra el algoritmo con el detalle de las transformaciones utilizadas en k NN Join en Spark.

Una mejora sobre H-BNLJ es construir una estructura de índice por cada bloque S , para ayudar a encontrar de forma más eficiente los vecinos de r en la misma cubeta [51]. La estructura seleccionada es un árbol Kd , la cuál organiza los puntos de un espacio euclidiano de k dimensiones. En la imagen 5.2 se ilustra como para la construcción del árbol se divide el plano con base en una dimensión y los puntos son divididos en subconjuntos. Cada nodo del árbol representa una lista de puntos contenida en una región.

Específicamente se optó por construir un árbol Kd dentro de cada cubeta, a partir de su bloque local S antes de obtener los vecinos. Dado un punto r el árbol es utilizado para obtener una lista de puntos ubicados en la misma región. Y con base en esta última lista se

Algoritmo 8: Etapa clasifica: Obtener vecinos más cercanos

```

Data: conjuntoS, conjuntoR, k
Result: clase
1 S ← conjuntoS.flatMap{ (id, instancia) →
2     /* 1. Dividir en bloques                                     */
3     bloque ← id%n
4     /* 2. Copiar cada bloque 'n' veces                           */
5     particiones.map{ particion →
6         cubeta ← asignarCubeta(particion,bloque,n)
7         (cubeta, (id,instancia))                                // Emitir.
8     }
9 }
10 R ← repetir pasos anteriores para asignar cubeta las instancias de R.
11 /* 3. Colocar en cubetas parejas de particiones.                */
12 cubetas ← S.cogroup(R)                                       // Join sobre bloques S y R.
13 /* 4. Obtener k-vecinos locales a cubetas.                     */
14 distancias ← cubetas.mapValues{ (bloqueS, bloqueR) →
15     distLocales ← bloqueR.flatMap{ (idr,instanciaR →
16         idr_dist = bloqueS.map{ (ids,instanciaS) →
17             dist ← distancia(idr, ids, metrica)
18             (idr, (ids, dist))
19         }
20         idr_locales = obtenerVecinos(idr_dist,k)
21         (idr_locales)
22     }
23     (distLocales)                                             // Emitir.
24 }
25 /* 5. Obtener resultado de knnJoin, vecinos de R sobre todo S. */
26 distRi ← distancias.flatMap{(cubeta,distancias)→
27     /* Cambiar la llave para agrupar por idr.                    */
28     distancias.map{(idr,(ids,dist)) → (idr,(ids, dist))}      // Emitir.
29 }.groupByKey()
30 /* 6. Por cada punto en R asignar clase con base en los vecinos. */
31 clasificacion ← distRi.map.({idr,distancias) →
32     kvecinos ← knn(distancias)
33     clase ← votacion(kvecinos)
34 }

```

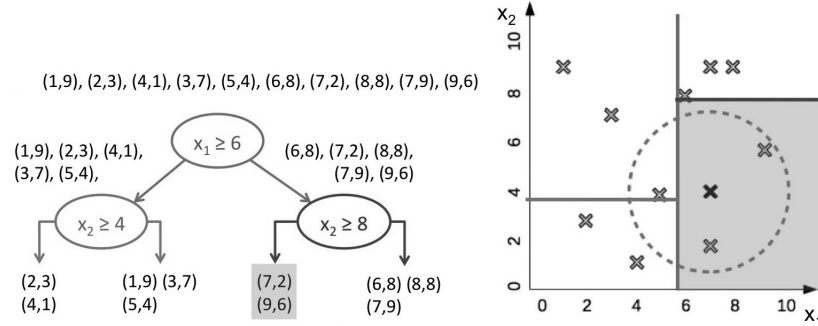


Figura 5.2: Ilustración de la construcción de un árbol Kd de dos dimensiones.

realizan los cálculos de distancias para obtener los k vecinos. Estos pasos se repiten por cada punto r en el bloque local R dentro de una cubeta. Es sencillo modificar el algoritmo 8 solo se incorporar la siguiente lógica en el paso 4 para considerar la construcción del árbol.

Algoritmo 9: Ajuste paso 4 de H-BNLJ en Spark para utilizar árbol Kd .

```

1 /* 4. Obtener k-vecinos locales a cubetas. */
2 distancias ← cubetas.mapValues{ (bloqueS, bloqueR) →
3     /* Construcción del árbol de bloque S es una cubeta. */
4     arbolKd ← contruyeArbol(bloqueS)
5     distLocales ← bloqueR.flatMap{ (idr, instanciaR) →
6         /* Obtener lista de puntos en la misma región que r. */
7         listaCercanos = arbolKd.recorre(r)
8         idr_dist = listaCercanos.map{ (ids, instanciaS) →
9             dist ← distancia(idr, ids, metrica)
10            (idr, (ids, dist))
11        }
12        idr_locales = obtenerVecinos(idr_dist, k)
13        (idr_locales)
14    }
15    (distLocales) // Emitir.
16 }
```

Una de las ventajas más evidentes de uso del árbol es la reducción de los cálculos de distancias a realizar, donde el costo adicional es la memoria requerida para mantener la estructura. El inconveniente que se puede presentar al utilizar dicha estructura es para aquellos puntos r que se encuentran en la frontera de una región, ya que es posible se descarten algunos puntos considerados vecinos ya que fueron colocados en otra. Para solucionar este inconveniente también se pueden consultar las regiones aledañas. Por lo que el uso del árbol resulta conveniente en k NN Join para reducir el tiempo de procesamiento.

Fue posible implementar k NN Join en Spark mediante RDD's y sus operaciones. Uno de

los principales problemas presentados fue el consumo de memoria y el tiempo de procesamiento. Durante la asignación de datos a cubetas, si estas contenían una cantidad de instancias, superiores a las capacidades de un nodo, se mostraba el error de consumo de memoria. Dicha situación, se resolvió reduciendo los elementos por partición, así como con el uso del árbol *Kd*. De esta configuración, un aspecto a considerar es, al definir más cubetas se incrementa el número de copias de instancias, sin embargo, a pesar de esto se logró dar continuidad al procesamiento.

5.3. Bosques Aleatorios en Spark

El método de Bosques Aleatorios es paralelizable de forma natural. Como se mencionó en la sección 3.2.2, este consiste en construir k árboles de decisión a partir del conjunto de entrenamiento. Donde la construcción de los árboles es independiente, por lo que se puede llevar a cabo de forma concurrente. Para clasificar una nueva instancia, cada árbol dentro del bosque la evalúa y emite un voto con su clase resultante, asignando la moda.

Un enfoque para realizar este procedimiento en Spark consiste en dividir en k bloques el conjunto de entrenamiento y por cada bloque construir un árbol. Este de forma automática asigna los bloques a los nodos dentro del *cluster* para realizar la construcción de los árboles de forma paralela. Específicamente, se requieren dos RDD's: uno para el conjunto de entrenamiento y otro para el conjunto de pruebas. En el RDD del conjunto de entrenamiento cada elemento equivalente a una instancia, a la que se le asignado un bloque, lo que produce la pareja `(id_bloque, instancia)`. Posteriormente se agrupan los elementos por llave generando un nuevo RDD `(id_bloque, lista(v))`, donde cada elemento corresponde a un bloque. Siguiendo, por cada bloque se construye un árbol utilizando la metodología seleccionada teniendo como resultado un RDD con el formato `(id_bloque, arbol)`, es posible notar que cada registro de este mantiene en memoria una estructura de árbol por cada bloque. Por último, la clasificación de una instancia de prueba con la forma `(id, instancia)` es evaluada por los árboles y obtiene el resultado final `(id, clase)`.

A continuación se escriben los pasos generales, seguido del detalle con las detalle utilizadas.

1. Dividir el conjunto de entrenamiento D en k bloques, donde k es la cantidad de árboles en el bosque.
2. Construir un árbol por cada bloque. Se utilizó la metodología: ID3 (explicada en la sección 3.2.1) y en caso de tratarse de valores continuos se utilizó el método de QUEST [27] para calcular el punto d de separación.
3. Para realizar la clasificación del conjunto de pruebas, por cada instancia los árboles dentro del bosque emiten un voto para obtener la clase.

Existe una implementación de Bosques Aleatorios dentro de MLlib, sin embargo se optó por desarrollar el procedimiento por diversas razones. Una de éstas era describir de forma detallada una alternativa para realizar clasificación mediante Bosques Aleatorios. Ya que en la documentación de esta, no es clara la metodología utilizada.

Algoritmo 10: Bosques Aleatorios

Data: k: cantidad de árboles,
 entrena: RDD del conjunto de entrenamiento,
 clasifica: RDD del conjunto de pruebas

Result: Conjunto de pruebas con su respectiva clase

```

1 /* 1. Dividir el conjunto de entrenamiento en k bloques. */
2 bloques ← entrena.map.{(id,registro) →
3     (id % k, registro)
4 }.groupByKey()
5 /* 2. Construir un árbol de decisión por cada bloque. */
6 bosques ← bloques.map{(id_bloque,lista) →
7     arbol ← crearArbol(lista,etiquetas)
8     (arbol) // Emitir.
9 }.collect()
10 /* 3. Realizar la clasificación del conjunto de pruebas. */
11 clasificacion ← clasifica.map{(registro) →
12     voto = bosques.map{ arbol →
13         (voto) // Emitir.
14 }

```

5.4. Apriori en Spark

En la sección 3.6.2 se revisó el algoritmo Apriori, el cuál es utilizado para encontrar conjuntos de elementos que comúnmente ocurren juntos. El método obtiene estos recorriendo una base de datos múltiples veces y generando conjuntos candidatos. Hace uso del soporte, medida que equivale al porcentaje de transacciones que contienen dichos conjuntos. Apriori descarta aquellos cuyo soporte no supere un umbral dado, devolviendo una lista con los que cumplan esta condición.

La implementación en Spark de Apriori se basa en una versión de MapReduce de [25]. Que consiste en dividir la base de datos, en este caso, el conjunto de entrenamiento D en bloques y por cada bloque realizar los conteos locales de los conjuntos. Posteriormente resumir los conteos locales para obtener el total global de cada conjunto.

Uno de los inconvenientes que presenta el uso de MapReduce para este algoritmo, se encuentra en que se deben ejecutar tantos trabajos como recorridos a la base de datos se realicen. Lo que implica que se debe cargar los registros de la base desde a disco para realizarlos, impactando en los tiempos para obtener resultados. Una de las ventajas de utilizar Spark se encuentra en que permite persistir en memoria dicha base de datos para acceder a ella de forma iterativa.

Llevar a cabo el procedimiento en Spark radica en crear un RDD_{BD} con las transacciones de la base de datos con la forma de la pareja $(id, transaccion)$ y mantener dicho RDD_{BD} en memoria para facilitar el acceso para futuras consultas. RDD_{BD} se divide en particiones

por Spark y por cada una se realizan los conteos de los conjuntos, para lo que se recorre la respectiva partición. Por cada transacción se obtiene una lista de los conjuntos que ocurren en esta, en esta a lista se emite la pareja (`conjunto`, 1) posteriormente se reducen las parejas y se obtienen los conteos globales. A partir de estos conteos se realiza el cálculo del soporte para seleccionar aquellos que superen el umbral. Siguiendo, con base en los conjuntos frecuentes de tamaño k se generan los candidatos de tamaño $k+1$, dicha lista es enviada a los nodos para recorrer la partición que mantienen de RDD_{BD} y realizar los conteos de los nuevos candidatos. El proceso continúa hasta que ya no se generen más conjuntos candidatos.

Los pasos para realizar Apriori en Spark se explica a continuación, seguido de las transformaciones específicas para realizar dichos pasos.

1. Crear RDD_{BD} y materializar en memoria.
2. Generar los conjuntos candidatos de tamaño $C_{k=1}$.
3. Obtener los conjuntos frecuentes $L_{k=1}$.
4. Generar conjuntos candidatos C_{k+1} . La generación de candidatos es la misma función que la usada por Apriori[1].
5. Escanear RDD_{BD} para obtener ocurrencia de los conjuntos de C_{k+1} . Con base en los conteos calcular su umbral y obtener L_k . Para esto Spark envía la lista C_{k+1} a cada nodo y estos últimos realizan el recorrido a la partición local que mantienen. Siguiendo se calculan suman los conteos de forma global. Por último se descartan aquellos que no superen el umbral especificado.
6. Repetir los pasos 4 y 5. Hasta que L_k resulte vacío, lo que significa que ningún conjunto supera el umbral.

Para encontrar conjuntos de elementos frecuentes MLib ofrece una implementación del algoritmo FP-Growth. Este último a diferencia de Apriori no utiliza la generación de candidatos, más bien hace uso de un árbol *fp* que permite guardar la ocurrencia de los conjuntos. Una de las mejoras que ofrece es que solo requiere recorrer la base de datos dos veces para la construcción del árbol. Específicamente la implementación de MLib es la versión paralela de FP-Growth denominada PFP [24] en MapReduce, en donde se construyen árboles a partir de grupos de transacciones. Un grupo, es a una porción de elementos de L_1 . Los grupos de transacciones, consisten en transacciones que han sido procesadas para asociarlas a un grupo. Esta separación permite separar mejor los datos para distribuir la construcción de los árboles de forma más eficiente en el modelo MapReduce.

La implementación de Apriori en Spark resulta sencilla. A diferencia de FP-Growth no requiere de la construcción de una estructura compleja. Spark ofrece el beneficio de realizar los conteos de forma paralela sobre bloques pequeños de transacciones. Además que los datos permanezcan en memoria facilita el acceso para los recorridos múltiples.

Algoritmo 11: Apriori en Spark

Data: DB, umbral

Result: L_k

```

1 /* 1. Crear RDDBD y persistir en memoria. */
2 transacciones ← sc.textFile(DB).persist
3 /* 2. Generar los conjuntos candidatos Ck=1 */
4 Ck=1 ← transacciones.flatMap{(id,elementos) →
5     elementos.map( elemento → (elemento,1)) // Emitir.
6 }.reduceByKey(_+_ )
7 /* 3. Obtener los conjuntos Lk=1 */
8 Lk=1 ← {elementos frecuentes k=1}
9 /* 4. Escanear RDDBD para obtener ocurrencia de los conjuntos de Ck */
10 for k ← 2 to Lk-1 do
11     /* 5. Generar conjuntos candidatos Ck+1 */
12     c ← apriori-gen(Lk-1)
13     conteos ← transacciones.flatMap{(id,trans) →
14         Ct ← subconjunto (Ck,t)
15         Ct.map(conjunto → (conjunto,1) // Emitir.
16     }.reduceByKey(_+_ )
17     Lk ← conteos.filter{ (conjunto,cantidad) →
18         (cantidad/numTrans) ≥ umbral
19     }
end for

```

5.5. k -Medias en Spark

El último algoritmo que se implementó en Spark fue k -Medias. Como se explicó en la sección 3.5.1, permite separar los datos en k grupos acorde a sus características. Para realizar esta separación hace uso de un representante de grupo, el cuál es utilizado para determinar la pertenencia a un grupo. Esto es, una instancia x se coloca en el grupo cuyo representante sea el más similar a x . Recordando que la métrica por defecto utilizada por k -Medias es la distancias Euclidianas. La cuál se utiliza para asignar un punto x al centro con la menor distancia.

El procedimiento de k -Medias es sencillo. Se inicia por seleccionar k centros de manera aleatoria, posteriormente se procede a calcular la distancias de las instancias del conjunto de entrenamiento a cada centro. Siguiendo, se procede a actualizar los centros calculando la media a partir de sus miembros. Se repite la asignación y actualización de centros hasta que no existan cambios. El resultado de k -Medias son los grupos con los respectivos representantes.

Dentro de k -Medias los cálculos más intensivos se realizan para las operaciones de: (a) asignación de grupo y (b) actualización de los centros de grupo, las cuáles se pueden realizar de forma paralela debido a que las instancias son independientes. De forma similar las sumas requeridas para el cálculo de los promedios de los centros. En Spark es posible llevar a cabo dichos cálculos por los nodos de un *cluster* de forma paralela mediante los RDD's y transformaciones.

Un enfoque para realizar a cabo k -Medias en Spark es el siguiente. Como se ha mencionado anteriormente, en Spark el procesamiento de los datos se lleva a cabo por bloques de datos (particiones) que son asignadas a los nodos del *cluster* para ser operados. Todas las implementaciones realizadas en Spark ha considerado esta condición. En k -Medias también se considera que el conjunto de entrenamiento se carga en memoria mediante particiones. Por lo que, el procedimiento inicia por crear k centros de manera aleatoria y enviar a cada nodo la lista con los centros. Así de forma local y paralela cada nodo calcula las distancias de sus instancias locales y realiza la asignación. Para llevar a cabo la actualización de centros, se cuenta forma local y se emiten los resultados parciales al programa principal, donde se suman los totales para calcular el promedio. Se repiten los pasos de enviar los centros actualizados a los nodos para repetir el proceso de asignación de centro, esto se itera hasta que no exista cambio en los centros.

Una de las ventajas que ofrece Spark para realizar la implementación de k -Medias es que permite mantener los datos en memoria. Se debe recorrer el conjunto de entrenamiento tantas veces como haya cambio en los centros, que los datos permanezcan en memoria hace más eficiente su acceso. Adicionalmente se puede controlar el número de particiones, indicado como parámetro el número de bloques en que se divide el conjunto de entrenamiento. Esto último ayuda a la escalabilidad, donde si la cantidad de particiones establecidas por Spark es menor a la cantidad de nodos disponibles, el usuario puede redefinir el número de particiones para asignar trabajo a mayor cantidad de nodos.

De forma específica para llevar a cabo una implementación de k -Medias en Spark se requiere de un $RDD_{entrena}$, con las instancias del conjunto de entrenamiento. En este caso,

el $RDD_{entrena}$ utiliza el formato $(id, (grupo, instancia))$, en donde por cada iteración durante la asignación de grupo el valor de `grupo` cambiará. Para llevar a cabo la actualización de los grupos, se emite `lista(grupo, vector_suma)`. Dentro de cada partición de forma local se realizan las sumas de las instancias correspondientes a cada grupo. Por ejemplo, si se tuvieran tres grupos, se emite una lista con tres vectores donde cada vector representa la suma parcial de las instancias por grupo. Así se envía una menor cantidad de valores al programa principal, que se encarga de sumar todos los resultados. Para obtener la media de los centros, se requiere contar el total de elementos por grupo, esto resulta sencillo, utilizando $RDD_{entrena}$ se emite $(grupo, 1)$. Posteriormente al utilizar la operación reducción y se obtiene $(grupo, cantidad)$. A partir de las sumas y cantidades recolectadas se obtienen los nuevos centros.

Durante la implementación de *k*-Medias se observó que un enfoque que permite mejorar los tiempos es realizar la mayor cantidad de operaciones de forma local y emitir al programa principal resultados parciales. Esto reduce la serialización y transmisión de los datos. Esto se pudo observar especialmente al momento de realizar las sumas para los promedios, primeramente se optó por descomponer las instancias en $((grupo, columna), valor)$ y utilizar la transformación *reduceByKey*. Pero utilizar la transformación *mapPartitions* requirió menos tiempo; ya que se accede a una partición completa, se realizan las sumas por grupo y se emiten *k* vectores de salida. Por ejemplo se pueden tener 100,000 elementos por partición con $k=3$, el resultado son *k* elementos con las sumas parciales por grupo. Se podría interpretar como una operación de reducción sobre vectores. En una prueba realizada mediante la primera opción el tiempo requerido fue 43mins, 13sec que se redujo a 10mins, 26sec en la segunda opción.

A continuación se muestran los pasos generales, seguido del detalle con las transformaciones.

1. Realizar particiones del conjunto de entrenamiento.
2. Calcular los centros aleatorios.
3. Asignar a cada instancia un grupo.
4. Actualizar los centros con base a los miembros. Los centros consisten en la media de los elementos que se encuentran en cada grupo.
 - a) Realizar sumas de los valores de cada atributo, de los elementos por grupo. Las sumas se realizan de forma local en cada partición.
 - b) Contar cantidad de elementos por grupo.
 - c) Obtener el promedio por grupo de (a) y (b)
5. Repetir los pasos 3 y 4 hasta que los centros no se modifiquen.

Fue posible realizar la implementación de *k*-Medias en Spark. En MLlib se ofrece la implementación paralela de *k*-Medis++ denominada *k*-Medias||. Una ventaja de desarrollar el método es que permite describir de forma específica una forma sencilla de implementar dicho método.

Algoritmo 12: k -Medias en Spark

Data: entrena: RDD del conjunto de entrenamiento, k Centros, particiones**Result:** instancias de entrena con grupo.

```

1 /* 1. Realizar particiones del conjunto de entrenamiento. */
2 entrena ← entrena.partitionBy(HashPartitioner(particiones)).persist
3 /* 2. Calcular los centros aleatorios. */
4 centrosIniciales ← centrosAleatorios( $k$ Centros, entrena)
5 esCentroDiferente ← verdadero
6 centros ← centrosIniciales
7 while esCentroDiferente do
8     /* 3. Asignar a cada instancia un grupo. */
9     entrenaGrupo ← entrena.map{ (id,(grupo,instancia)) →
10         nuevoGrupo ← asignaCentro(instancia,centros)
11         (id, (nuevoGrupo,instancia)) // Emitir.
12     }
13     /* 4. Actualizar los centros con base a los miembros. */
14     centrosNuevos ← actualizarCentro(entrenaGrupo)
15     esCentroDiferente ← cambioCentro(centros,centrosNuevos)
16     centro ← centrosNuevos
17     /* 5. Repetir los pasos 3 y 4 hasta que los centros no se modifiquen. */
end while

```

5.6. Resumen

En el presente capítulo se mostró de forma detallada la implementación de cinco algoritmos de aprendizaje en Spark. La versión tradicional de estos se describió en el capítulo 3 y los métodos abordados la siguiente:

Los métodos son diferentes sin embargo una característica en común es que son de naturaleza paralela. Esto ayudó a expresar de forma más sencilla los métodos en las operaciones: *map*, *reduceByKey*, etc. Considerando su naturaleza, el enfoque fue dividir los datos en particiones (bloques) y realizar las operaciones de forma local por bloque y al finalizar sumarizar los resultados parciales para obtener el resultado final. Como se describe a continuación de forma resumida por cada método:

- Bayes Ingenuo: realizar los conteos de histogramas por partición.
 - Bosques Aleatorio: construir cada árbol por cada partición.
 - k NN Join: asignar una pareja de bloques R y S a cada partición. En este caso se observó que es mejor tener una distribución uniforme, donde se recomienda cada cubeta tenga la misma cantidad de datos. Debido a que si una partición tiene más elementos se consume la memoria. Fue el método al que se le asignó la mayor cantidad de particiones para tener una cantidad de instancias que fuera posible procesar por los nodos.
 - k -Medias: se obtiene la media del grupo por partición.
 - Apriori: se lleva a cabo el conteo de elementos de forma local. A diferencia de k NN Join, se prefiere tener una cantidad menor de particiones ya que se emiten los elementos diferentes por partición. Donde a mayor cantidad de particiones mayor la transferencia de elementos por red, siendo que se busca reducir la transferencia.
-

Capítulo 6

Bases de datos

A continuación se describen las bases de datos utilizadas para realizar los experimentos. Fueron seleccionadas cuatro bases de datos que cumplieran con características de Big Data. Se muestra el detalle de cada base de datos desde cantidad de registros, estructura, distribución de clases, etc.

En los cuatro casos fue necesario dar el formato adecuado a las bases de datos para llevar a cabo las pruebas de los métodos implementados. Por lo que se utilizó la metodología KDD revisada en la sección 2.1. Para facilitar la lectura a continuación se lista nuevamente los pasos que comprende el método.

1. Limpieza
2. Selección
3. Procesamiento Previo
4. Transformación
5. Minería de Datos
6. Interpretación y Evaluación

6.1. Enron

La primer base de datos es una colección de correos electrónicos de empleados de la corporación Enron. Se hizo pública por la Comisión Federal Reguladora de Energía (*Federal Energy Regulatory Commission*) durante la investigación realizada posterior al cierre de esta. Al tratarse de una base de datos real y pública ha sido adoptada como punto de referencia para la evaluación de métodos como clasificación de texto entre otros. Motivado por el uso práctico, para propósitos de investigación es posible encontrarla etiquetada con las categorías: “valido” (*ham*) e “invalido” (*spam*). Dicha versión se encuentra disponible para descarga de [9].

Enron cuenta con un total de 33,716 correos. De los cuales 16,545 corresponden a la categoría “valido” (de interés para el usuario) y 17,171 a “invalido” (sin contenido interesante

para el usuario). El peso de total de los correos equivale a 53.9 MB. El *corpus* se encuentra disponible en 6 carpetas., su distribución por carpeta se muestra en la tabla 6.1, junto con la proporción por cada categoría.

Archivo	Validos	Inválidos	Total	Proporción
1	3,672	1,500	5,172	3-1
2	4,361	1,496	5,857	3-1
3	4,012	1,500	5,512	3-1
4	1,500	4,500	6,000	1-3
5	1,500	3,675	5,175	1-3
6	1,500	4,500	6,000	1-3
Total:	16,545	17,171	33,716	

Cuadro 6.1: Descripción Enron

Al tratarse de una base de datos de texto corresponde al tipo de dato no-estructurado. Los métodos implementados requieren un formato estructurado para operar, por lo que fue necesario procesar el conjunto al formato adecuado. A continuación se muestran que se llevaron a cabo para dar el formato requerido para los algoritmos:

- Selección:** La representación seleccionada fue una matriz de frecuencia de términos. Donde una fila representa un documento y una columna una palabra, el valor de una posición a_{ij} , (documento i en la columna j) corresponde a la cantidad de veces que ocurre una la palabra de la columna a_j en el documento i . Esta representación se basa en un vocabulario, el cuál se conforma de las palabras distintas que ocurren dentro del *corpus*. Estas palabras equivalen las columnas de la matriz. En los documentos de texto es posible encontrar una gran cantidad de palabras que no aportan información, sin embargo si pueden impactar negativamente en los resultados así como los tiempos de procesamiento. Por lo que se realizó este paso, para seleccionar ciertas características sin afectar los resultados y obtener mejores tiempos de respuesta en comparación si se utilizan todas.

Una forma de selección consiste en evaluar las características y solo aquellas cuyo resultado supere un umbral dado son consideradas informativas. Una métrica de evaluación sencilla y que ha presentado buenos resultados, es la frecuencia de documentos [49]. Que consiste en contar la cantidad de ocurrencias de una palabra dentro del *corpus*, aquellas palabras cuya frecuencia no supere el umbral dado se descartan. Mediante esté enfoque es posible eliminar hasta el 90 % de los términos sin afectar los resultados drásticamente. Debido a que este criterio es sencillo de utilizar y eficiente fue utilizado para realizar la selección. Se eligieron aquellas cuya frecuencia por clase superara un umbral de 0.1 % para construir el vocabulario.

- Transformación:** A partir del vocabulario es posible transformar los textos en una matriz de frecuencia de términos. Para esto se cuenta la frecuencia de los términos del

vocabulario dentro de cada documento i y asignar ese valor en la posición correspondiente j . Esta representación fue utilizada por casi todos los métodos con excepción de Bayes Ingenuo y Apriori.

- **Minería de Datos:** Los métodos que fueron evaluados: Bayes Ingenuo, Bosques Aleatorios, k NN, Apriori, Fp-Growth.

La clasificación de texto es una tarea que cada vez adquiere mayor presencia. Se ha mencionado anteriormente que mediante la adopción de dispositivos electrónicos resulta más sencilla la digitalización de datos. El texto es uno de los datos que continúan aumentando su producción. Por ejemplo, hoy en día es cada vez más necesario contar con una cuenta de correo electrónico. Gran cantidad de contenido web es texto. Inclusive ya es posible adquirir libros en formato digital. En estos son ejemplos los algoritmos de minería como clasificación o agrupación pueden ser utilizados.

Como es posible notar ENRON es una base de datos relativamente pequeña. Sin embargo fue seleccionada como punto de partida para el desarrollo de los métodos en Spark y realizar pruebas con un volumen manejable. Posteriormente se seleccionaron bases con mayor tamaño.

6.2. *Record Linkage Comparison Patterns*

Esta base de datos es una muestra del registro epidemiológico del cáncer de Alemania. Los registros contienen información personal de pacientes recolectados a los largo de tres años en el período de 2005 a 2008. Cada instancia representa un patrón de comparación que tiene asociada una de dos posibles clases: “coincide (*TRUE*)” y “no-coincide (*FALSE*)”. Es posible descargarla del repositorio de UCI [40].

La base de datos contiene 5,749,132 registros de los cuales 20,931 corresponden a la clase “coincide” y 5,728,201 a “no-coincide”. La base tiene un peso de 252.5 megabytes. La componen 12 atributos, de los cuáles son: 9 predictivos, 2 no predictivos y 1 clase. En este caso, dicha base incluye campos con falta de valores que se indican con el símbolo ‘?’’. Se encuentra en 10 archivos de texto separados por coma (.csv). En la tabla 6.2 se muestra la distribución de los archivos, donde se observa que existe un evidente desproporción entre las dos clases, con una predominancia de instancias de la clase “no-coincide”. Esta condición debe ser considerada para los métodos de evaluación seleccionados.

- **Procesamiento Previo:** RLCP se encuentra en un formato estructurado. Sin embargo, es necesario manejar la falta de valores, ya que se requiere reemplazar el símbolos ‘?’’ por un valor real, para los métodos que calculan la distancia Euclidiana como k NN y k -Medias, así como para el calculo de probabilidades de Bayes Ingenuo. En este caso la tarea que se realizó fue sustituir el símbolo ‘?’’ por la media de los valores de cada atributo correspondiente.
 - **Minería de Datos:** Los métodos que fueron evaluados: Bayes Ingenuo, Bosques Aleatorios, k NN, k -Medias.
-

Archivo	Validos	Inválidos	Total
1	572,820	2,093	574,913
2	572,820	2,093	574,913
3	572,821	2,093	574,914
4	572,820	2,093	574,913
5	572,820	2,093	574,913
6	572,820	2,093	574,913
7	572,820	2,093	574,913
8	572,820	2,093	574,913
9	572,820	2,093	574,913
10	572,820	2,094	574,914
Total:	5,728,201	20,931	5,749,132

Cuadro 6.2: Distribución *Record Linkage Comparison Patterns (RLCP)*

RLCP contiene una mayor cantidad de registros en comparación con Enron, llevar a cabo tareas de minería sobre una base de aproximadamente 5 millones empieza a ser un buen punto de partida para las pruebas.

6.3. Hepmass

Se trata de una base de datos del dominio de la física. Específicamente de Física de alta energía, campo se encarga de estudiar las partículas que componen la materia, la energía y las interacciones entre ellas. Hepmass fue generada a partir de simulaciones Monte Carlo de colisiones de partículas. Cada instancia representa una partícula asociada a un tipo: “señal” y “fondo”. La aplicación de interés se encuentra en detectar partículas de tipo “señal” debido a que estudiarlas puede llevar a encontrar información crítica sobre la naturaleza de la materia. Esta base es publica y que se encuentra disponible en [15].

Hepmass contiene 10,500,000 millones de registros de partículas, de los cuales 5,250,124 pertenecen a la clase “señal” y 5,249,876 corresponden a “fondo”. Presenta 28 atributos: 1 para la clase, seguido de 27 características descriptivas (22 características de bajo nivel y 5 de alto nivel) y la masa de la partícula. Tiene un peso total de 7.6 GB. Se encuentra en dos archivos en formato CSV, uno con 7.0 y otro con 3.5 millones respectivamente. Para llevar a cabo las prueba, se separaron en 10 bloques, cuya distribución de datos se muestra en la tabla 6.3. Es posible notar que la proporción de las clases es más equitativa en comparación con RLCP.

Debido a que Hepmass es una base de datos estructurada con únicamente valores continuos y no presentaba falta de valores. Solo se llevó a cabo la separación de los archivos fuente, como se describe a continuación:

- **Procesamiento Previo:** Se procedió a separar en 10 bloques los dos archivos origina-

Bloque	“Fondo”	“Señal”	Total
1	499,421	500,579	1,000,000
2	500,408	499,592	1,000,000
3	499,585	500,415	1,000,000
4	499,786	500,214	1,000,000
5	500,180	499,820	1,000,000
6	499,806	500,194	1,000,000
7	499,935	500,065	1,000,000
8	500,102	499,898	1,000,000
9	499,944	500,056	1,000,000
10	750,709	749,291	1,500,000
Total	5,249,876	5,250,124	10,500,000

Cuadro 6.3: Cantidades de registros de Hepmass

les. La división de los bloques fue de forma secuencial, esto es, se tomo el archivo con 7 millones de registros y respetando el orden de los registros, se fueron separando en bloques consecutivos. Lo mismo se realizó con el segundo archivo.

Adicionalmente se asignó un identificador único. Esto para el procedimiento de k NN Join con el propósito de tener una distribución uniforme sobre la distribución de los datos así como control sobre esta.

- **Minería de Datos:** los métodos evaluados son: Bayes Ingenuo, Bosques Aleatorios, k NN y k -Medias.

Una consideración de Hepmass es que las cuatro base de datos fue la de mayor tamaño.

6.4. Reuters RCV1

La última bases de datos utilizada para pruebas fue Reuters RCV1, la cuál es una colección de noticias proporcionada por la agencia internacional de noticias Reuters, que publica las historias vía Internet. En 2004 liberaron para propósitos de investigación, un corpus de aproximadamente 800,000 noticias en idioma inglés denominadas Reuters RCV1. Las cuáles cubren un año de publicaciones entre 20/Agosto/1996 a 19/Agosto/1997. También se trata de una base publica, sin embargo, se requiere solicitar el acceso para la descarga en [37]. En dicho sitio se encuentran tres conjuntos disponibles:

- RCV1: Reuters Corpus, Volumen 1, contiene 800,000 noticias en inglés. Este requiere 2.5 GB de almacenamiento después de descomprimir el archivo.
- RCV2: Reuters Corpus, Volumen 2 con 487,000 noticias en 13 idiomas diferentes.

- TRC2: 1,800,370 notas en el período 01/Enero/2008 a 28/Febrero/2009 inicialmente proporcionado en la conferencia *Text Retrieval Conference (TREC)*. Con un peso 2.6 GB de almacenamiento posterior a descomprimir el archivo.

De los tres corpus solo RCV1 y RCV2 se encuentran etiquetados. Pero RCV1 fue etiquetado de forma manual por el personal de Reuters y contiene mayor cantidad de documentos. En [23] se indican las políticas de etiquetado del *corpus* con detalle. Pero de forma general, cada noticia tiene como propiedades tres categorías, que le son asignadas códigos de tres conjuntos: Tópicos, Industrias y Regiones. Como regla todas las notas tienen asignadas al menos un código por categoría. En donde para propósitos de clasificación los códigos pueden ser utilizados como clases.

Las noticias se encuentran en formato XML, un ejemplo de documento se muestra en la figura 6.1. En dicho esquema las etiquetas de `<codes></codes>` son las categorías. Además es posible observar que tres propiedades son independientes, y sus valores corresponden a los códigos de los respectivos conjuntos. En el ejemplo, la categoría Tópicos tiene asignados 5 códigos. Es importante mencionar que existe una jerarquía dentro de los códigos. Esta condición es considerada en la segunda regla de etiquetado que indica que al asignarse un código también se agregan sus ancestros. En el ejemplo: el código general es “CCAT” y sus descendientes directos son: “C15” y “C18”, que a su vez son predecesores de “C152” y “C181”.

Reuters RCV1 solo es una colección de noticias y no contiene un formato específico para realizar pruebas de minería. Sin embargo, al tratarse de un *corpus* etiquetado puede ser utilizado para diversas aplicaciones. Por sus características puede ser útil para clasificación jerárquica o multi-clase (a diferencia de clasificación binaria donde solo se tienen dos categorías, en este tipo se consideran n categorías). RCV1 resulta versátil, ya que permite dar el formato adecuado para diversas aplicaciones de minería.

En este trabajo se optó por realizar clasificación multi-clase. Para dicha aplicación se limitó a utilizar únicamente la categoría de tópicos. Donde se identificaron cuatro códigos principales: “CCAT” (Corporativo/Industrial), “ECAT” (Economía), “GCAT” (Gobierno/Social) y “MCAT” (Mercados). Específicamente la aplicación consiste en clasificar una nota a una de estas cuatro categorías. Dadas estas restricciones se tuvo que seleccionar las notas que únicamente tuvieran asociado uno de los cuatro código, el resultado se referirá como Reuters_4CAT, para diferenciarlo del *corpus* original. Cabe mencionar que del conjunto original se descartaron 179,554 que no satisfacían el criterio de selección.

Reuters_4CAT contiene 682,147 instancias, de los cuáles: 15,233 pertenecen a la categoría “MCAT”, 152,020 a “GCAT”, 50,091 de “ECAT” y por último 273,077. Su peso es de 1.8 GB y se encuentra en cuatro carpetas, cada una de estas con tiene noticias de la misma clase. Sin embargo, para realizar pruebas se separaron las instancias por bloques, la distribución se muestra en la tabla 6.4

Debido a que Reuters es una base con datos no-estructurados se requieren llevar a tareas de procesamiento previo. De la misma forma que ENRON se debe transformar a una representación de matriz de frecuencia de términos. Sin embargo, para el caso de Reuters_4CAT este se encuentra en un formato XML de donde se interesa extraer tres campos: *itemid*, *texto*

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<newsitem itemid="2330" id="root" date="1996-08-20" xml:lang="en">
<title>USA: Tylan stock jumps; weighs sale of company.</title>
<headline>Tylan stock jumps; weighs sale of company.</headline>
<dateline>SAN DIEGO</dateline>
<text>
<p>The stock of Tylan General Inc. jumped Tuesday after the maker of
process-management equipment said it is exploring the sale of the
company and added that it has already received some inquiries from
potential buyers.</p>
<p>Tylan was up $2.50 to $12.75 in early trading on the Nasdaq market.</p>
<p>The company said it has set up a committee of directors to oversee
the sale and that Goldman, Sachs & Co. has been retained as its
financial adviser.</p>
</text>
<copyright>(c) Reuters Limited 1996</copyright>
<metadata>
<codes class="bip:countries:1.0">
<code code="USA"> </code>
</codes>
<codes class="bip:industries:1.0">
<code code="I34420"> </code>
</codes>
<codes class="bip:topics:1.0">
<code code="C15"> </code>
<code code="C152"> </code>
<code code="C18"> </code>
<code code="C181"> </code>
<code code="CCAT"> </code>
</codes>
<dc element="dc.publisher" value="Reuters Holdings Plc"/>
<dc element="dc.date.published" value="1996-08-20"/>
<dc element="dc.source" value="Reuters"/>
<dc element="dc.creator.location" value="SAN DIEGO"/>
<dc element="dc.creator.location.country.name" value="USA"/>
<dc element="dc.source" value="Reuters"/>
</metadata>
</newsitem>
```

Figura 6.1: Ejemplo de un documento del corpus Reuters RCV1

Bloque	“MCAT”	“GCAT”	“ECAT”	“CCAT”	Total
1	15,291	15,263	5,006	27,225	62,785
2	15,291	15,263	5,006	27,225	62,785
3	15,175	15,010	5,152	27,282	62,619
4	15,526	15,172	4,979	27,195	62,872
5	15,232	15,152	4,975	27,256	62,615
6	15,103	15,314	5,028	27,373	62,818
7	15,214	15,271	5,093	27,398	62,976
8	15,389	15,109	4,971	27,399	62,868
9	15,505	15,223	4,946	27,250	62,924
10	15,233	15,243	4,935	27,474	62,885
Total	152,959	152,020	50,091	273,077	628,147

Cuadro 6.4: Distribución de registros de Reuters_4CAT.

y *código*, y con base a estas construir el formato de instancia requerido. A continuación se describen las tareas dentro de *KDD* realizadas. Debido a que fueron varias las operaciones para obtener el formato deseado se repitieron algunos pasos.

- **Transformación (Ronda 1):** En esta etapa se leen los archivos en formato XML, de los cuales se extraen los campos: id de la noticia, texto, y el tipo de tópico codificado. Se genera una instancia con tres campos: id_noticia, texto y clase (representada por el código). En este mismo paso dentro del texto se eliminan: palabras paro y caracteres no alfanuméricos. Por palabras paro, se refiere aquellas que no aportan ninguna información y son comunes a todo el *corpus*, algunos ejemplo son: pronombres, artículos, conjunciones, etc.
- **Selección:** Como se mencionó en la descripción de ENRON, antes de construir la matriz de frecuencia de términos se quiere obtener el vocabulario. En este caso se seleccionaron por clase las palabras cuya frecuencia superará un umbral de 0.1%. Porcentaje que fue seleccionado de forma empírica, se realizaron pruebas con diferentes umbrales y dicho porcentaje presento los mejores resultados de precisión en tiempos aceptables.
- **Transformación (Ronda 2):** Con base al formato de instancias de la primera ronda de transformación y al vocabulario, se procedió a construir la matriz de frecuencia de términos.
- **Minería de Datos:** Para realizar clasificación, se ejecutaron los métodos: Bayes Ingenio, *k*NN, Bosques Aleatorios y *k*-Medias. Adicionalmente este conjunto se utilizó para realizar pruebas sobre los métodos de conjuntos frecuentes. Donde una noticia representa una transacción y las palabras los elementos. Donde se utilizó el vocabulario para obtener los elementos distintos de la base de datos. Los métodos evaluados son: Apriori y Fp-growth.

Capítulo 7

Experimentos

En este capítulo se describen los experimentos realizados sobre las cuatro bases de datos y los métodos implementados así como las versiones de MLib. Además se presenta: forma de evaluación, métricas seleccionadas y características de la plataforma utilizada.

7.1. Validación Cruzada

Es una técnica ampliamente utilizada para evaluar métodos de aprendizaje maquina, a partir de los resultados de predicción de un modelo. Consiste en dividir el conjunto de entrenamiento en k particiones, idealmente con la misma cantidad de elementos, de los cuales se seleccionan $k - 1$ bloques para construir el modelo, y el restante para evaluar dicho modelo. Tarea que se repite k veces, en cada ocasión se toma un bloque diferente para evaluación. Al finalizar se calcula la media aritmética de los k resultados obtenidos. En la imagen 7.1 se ilustra el método.

En este trabajo se seleccionó $k=10$, el cual es un valor típicamente seleccionado [21]. Razón por la que las bases de datos fueron separadas en 10 bloques, con excepción de ENRON que se presentaba en 6 bloques. Utilizando validación cruzada, se construyó el modelo con 9 y 5 bloques respectivamente reservando 1 bloque para pruebas. En total se llevaron a cabo 10 iteraciones alternando el bloque de pruebas.

Durante cada iteración se obtuvo la matriz de confusión, la cual es requerida para calcular las métricas de evaluación. Dicha matriz registra la relación de instancias que fueron clasificadas correctamente y las que no. En la imagen 7.1 se muestra la matriz de confusión para un ejemplo de dos clases. En donde se contraponen la clase real contra la predicción de las instancias de prueba. Los valores de la diagonal corresponden a aquellas que fueron clasificadas correctamente.

7.2. Métricas de evaluación

Para evaluar la precisión de los métodos seleccionados se eligieron 6 métricas, las cuales son ampliamente utilizadas en la literatura. Las cuales a partir de la matriz de confusión es posible calcularlas. Una de las más utilizadas es Exactitud, sin embargo debido a que presenta

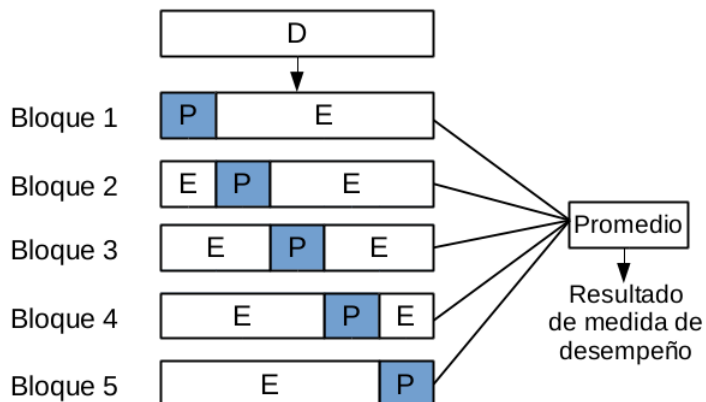


Figura 7.1: Validación cruzada con k bloques. Donde P=partición de pruebas, E=partición de entrenamiento.

	Predicción Clase=0	Predicción Clase=1
Real Clase=0	Verdaderos Positivos (tp) <i>correctos</i>	Falsos Negativos (fn) <i>incorrectos</i>
Real Clase=1	Falsos Positivos (fp) <i>incorrectos</i>	Verdaderos Negativos (tn) <i>correctos</i>

Cuadro 7.1: Matriz de confusión para clasificación binaria

únicamente la relación de instancias clasificadas correctamente y el total, no siempre muestra el panorama completo sobre las predicciones y la clases reales. Por lo que se seleccionaron otras 5 medidas seleccionadas, las cuáles se describen a continuación así como la fórmula para obtenerlas.

- Exactitud (*Accuracy*, *ACC*): Es la proporción de clasificaciones correctas entre el total de instancias.

$$Exactitud = \frac{tn + tp}{tp + tn + fp + fn}$$

- Precisión (*Precision*) o Valor Predictivo Positivo (*Positive Predictive value*, *PPV*): Con base en las predicciones de clase realizadas, calcula el porcentaje de estas que fueron correctas.

$$Precision = \frac{tp}{tp + fp}$$

- Memoria (*Recall*) o Tasa de verdaderos positivos (*True Positive Rate, TPR*): Considerando la clase real de una instancia. Determina que porcentaje de instancias que debieron tener la etiqueta x fueron correctas.

$$Memoria = \frac{tp}{tp + fn}$$

- Valor - F: Es la media aritmética de la precisión y memoria.

$$Valor - F = 2 \cdot \frac{(1 + \beta^2)(precision \cdot memoria)}{\beta^2 \cdot (precision + memoria)}$$

- Coeficiente de Correlación de Matthews (MCC): La medida de exactitud toma en cuenta la proporción de las instancias clasificadas correctamente, sin embargo es sensible cuando los conjuntos son de diferente tamaño. Por ejemplo, considere el caso de *RCLP*, que existe una proporción de 200-1 (por 1 instancia de la clase “coincide” existen 200 instancias de la contra parte). Al tener una tasa alta de altos positivos y una baja de verdaderos positivos, se obtendrían buenos resultados. MCC toma en consideración los verdaderos y falsos positivos, siendo una medida balanceada para casos donde la población entre la clases no es proporcional.

$$MCC = \frac{tp \times tn - fp \times fn}{\sqrt{(tp + fp)(tp + fn)(tn + fp)(tn + fn)}}$$

- Tasa de Falsos Postivos (*False Positive Rate, FPR*): Es la proporción de instancias que el modelo les asigno la clase “x” y en realidad pertenecían a la clase “x”.

$$FPR = \frac{fp}{fp + tp}$$

7.3. Equipo

Las pruebas se realizaron sobre un *cluster* tres nodos, con un sistema operativo Mac OS X. Los tres nodos se identifican con la siguiente nomenclatura así como la descripción de cada uno:

- *Slave* con 4 núcleos (*cores*), 8 GB de memoria RAM y 300 GB en disco duro.
- *Master* con 4 núcleos, 8 GB de memoria RAM y 500 GB en disco duro.
- *Capitan* con 8 núcleos, 32 GB de memoria RAM y 500 GB en disco duro.

En total se contaba con 48 GB en memoria, 16 núcleos y 1.3 TB para procesar de los datos. Recordando que en Spark la unidad de procesamiento son los procesos trabajadores (*executors*) a los cuales le son asignados recursos. La configuración seleccionada fue: 4 GB de memoria RAM y 2 núcleos de procesamiento, asignada a cada trabajador. A pesar de que Spark se encarga de la distribución de trabajo, también es posible especificar el número de trabajadores a utilizar.

7.4. Formato de pruebas

El procedimiento realizado para llevar a cabo las pruebas. Consta de los siguientes pasos:

1. Dividir los datos en dos conjuntos: entrena y clasifica
2. Entrenar el modelo por cada método.
3. Evaluar las instancias del conjunto clasifica, por cada modelo.
4. Obtener matriz de confusión y calcular las métrica definidas.
5. Repetir los pasos anteriores $k=10$ veces. Ya que se está utilizando validación cruzada.
6. Calcular el promedio de las medidas obtenidas.

Como se ha mencionado anteriormente, es posible categorizar los métodos en tres tipos de tareas:

- Clasificación: Bayes Ingenuo, Bosques Aleatorios y knn Join.
- Agrupación: k -Medias
- Conjuntos frecuentes: Apriori, Fp-Growth

Debido a que se trata de tareas distintas las métricas de evaluación son diferentes. Para no presentar un extenso escenario de pruebas, se utilizaron dos tipos de evaluación: clasificación y conjuntos frecuentes. Dentro de clasificación se utilizan las seis métricas mencionadas anteriormente. Donde además de los métodos de clasificación (Bayes, Bosques y kNN) en este tipo de pruebas también se incluyó a k -Medias. Aunque este último método no es para dicha tarea, para evaluarlo de esta forma, se realizó clasificación con base en los centros. Del segundo tipo se incluyen a los métodos de conjuntos frecuentes, donde se registraron los tiempos de respuesta y cantidad de conjuntos resultantes.

En la 7.2 se indica la relación de los métodos a evaluar con respecto a la base de datos utilizada.

Método	ENRON	RCLP	Hepmass	Reuters
Bayes Ingenuo	x	x	x	x
Bosques Aleatorios	x	x	x	x
kNN Join	x	x	x	x
k -Medias		x	x	
Conjuntos frecuentes	x			x

Cuadro 7.2: Métodos para ejecutar

7.5. Resultados

Las pruebas se realizaron utilizando las versiones implementadas así como en las versiones que ofrece MLlib, sobre las bases de datos descritas en la sección anterior. Los resultados se muestran en el mismo orden, en que se describieron las bases de datos.

Como se mencionó anteriormente, uno de los puntos de partida para las implementaciones realizadas fueron las versiones de los algoritmos en MapReduce. En el trabajo [28], presenta una implementación de: Bayes Ingenuo, Bosques Aleatorios y k NN Join en Hadoop. El cuál se llevo a cabo por una compañera de maestría de la UAM Iztapalapa, donde se utilizó la misma infraestructura y tres de las bases de datos (ENRON, RCLP y HEPMASS) utilizadas en este trabajo. Se incluyen los resultados reportados en dicho trabajo como métrica de comparación.

Para diferenciar entre los resultados de: implementaciones realizadas, MLlib y Hadoop; se asignaron nombres para identificarlos de la siguiente forma:

- Spark: versiones implementadas
- MLlib: implementación de MLlib
- Hadoop: implementaciones en Hadoop y MapReduce del trabajo [28].

Adicionalmente se muestran tres tipos de resultados. Los dos primeros son para los métodos de clasificación y el segundo para conjuntos frecuentes:

- (a) En la primer relación, se muestra el desempeño del modelo independiente a la clase, se incluyen las métricas: Exactitud, MCC y tiempo. El tiempo se refiere, al requerido para obtener los resultados, esto incluye entrenamiento y clasificación durante las diez validaciones cruzadas.
- (b) En la segunda relación se muestran el desempeño del modelo respecto a una clase, las métricas son: FPR, Memoria, Precisión, Medida - F.
- (c) La tercera relación es exclusiva para las bases de texto, donde se evaluaron los algoritmos de conjuntos frecuentes.

7.5.1. ENRON

Como se muestra en la tabla 7.3 y en la así como en la gráfica 7.2, la mejor Exactitud la obtuvo Bayes Ingenuo de Spark con 0.972, seguida de k NN Join de Hadoop con 0.915. De MCC el mejor resultado lo presentó Bayes Ingenuo de Spark con 0.931 seguido de k NN Join de Spark. Adicionalmente en la figura 7.3 se muestra, que el método con menores tiempos en presentar resultados fue Bayes Ingenuo de Spark mientras que Bosques Aleatorios de Spark fue el que tomó más tiempo.

Método	Versión	Exactitud	MCC	Tiempo
Bayes Ingenuo	Spark	0.972	0.931	04min, 12seg
	MLlib	0.889	0.751	05min, 35seg
	Hadoop	0.902	.-	01hr, 23min, 27seg
Bosques Aleatorios	Spark	0.881	0.722	11min, 52seg
	MLlib	0.806	0.589	05min, 16seg
	Hadoop	0.829	.-	.-
kNN Join	Spark	0.891	0.773	08min, 42seg
	Hadoop	0.915	.-	

Cuadro 7.3: Medidas de calidad de los métodos sobre ENRON

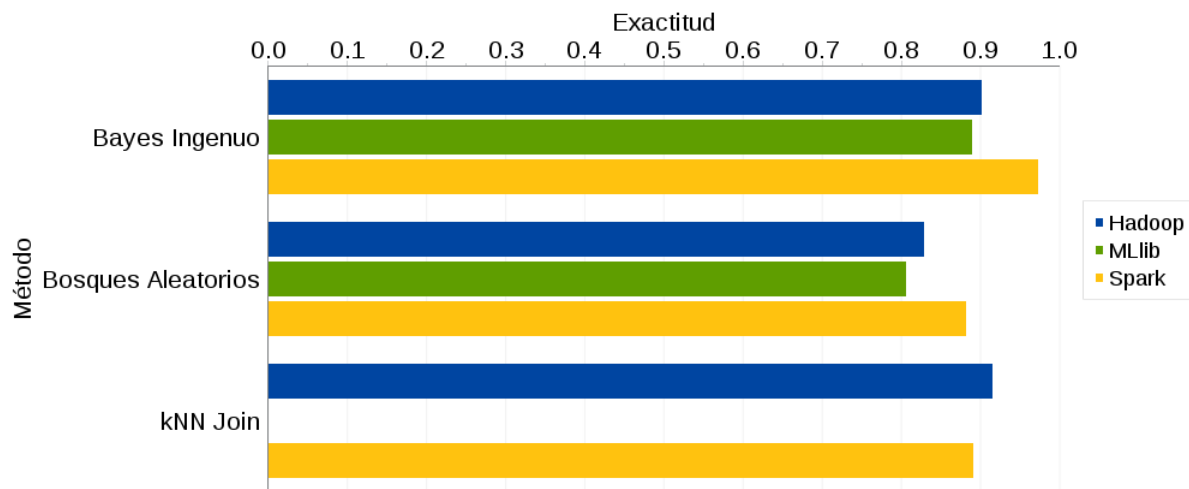


Figura 7.2: Resultados para la medida de desempeño de exactitud de ENRON.

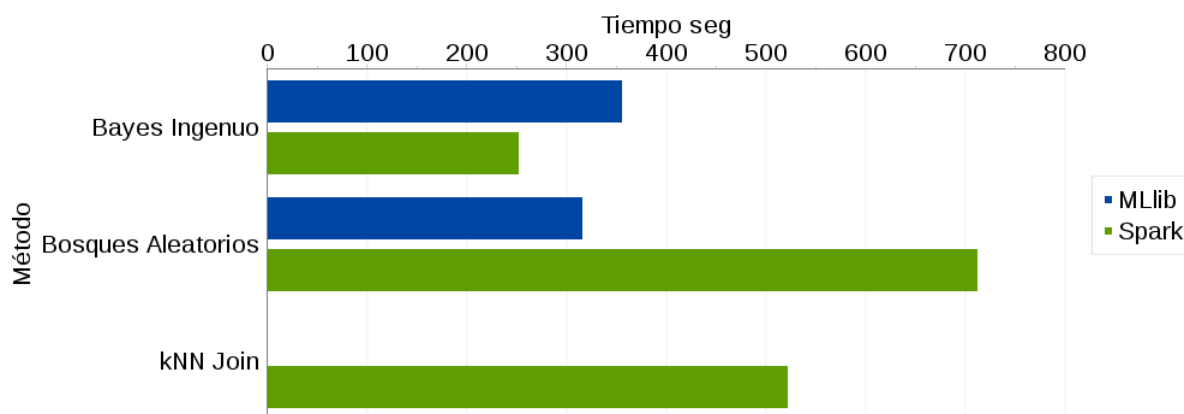


Figura 7.3: Resultados para los tiempos registrados para ENRON.

Método	Versión	FPR	Memoria	Precisión	Valor - F
Bayes Ingenuo	Spark	0.063	0.990	0.947	0.967
	MLib	0.157	0.930	0.888	0.903
	Hadoop	.-	0.904	0.839	0.868
Bosques Aleatorios	Spark	0.172	0.934	0.816	0.864
	MLib	0.401	0.978	0.694	0.799
	Hadoop	.-	0.745	0.935	0.815
k NN Join	Spark	0.175	0.980	0.793	0.866
	Hadoop	.-	0.956	0.854	0.901

Cuadro 7.4: Detalle de medidas de la clase = 'Spam'

De los procedimientos Bayes Ingenuo fue el que presentó los mejores resultados. Se ha reportado con anterioridad que Bayes Ingenuo es un buen clasificador en colecciones de texto. Adicionalmente fue el que requirió menos tiempo.

La segunda tabla de resultados 7.4 contiene los resultados asociados a la clase "Spam". Para la tasa de falsos positivos FPR, lo mejores resultados, esto es, aquellos con la menor tasa; fue para Bayes Ingenuo de Spark seguida de Bayes Ingenuo de MLib. En el caso de la memoria los resultados más altos los presentó Bayes Ingenuo Spark con 0.990. En precisión Bayes Ingenuo de Spark obtuvo los mejores con 0.947. Por último, en Valor - F, Bayes Ingenuo de Spark obtuvo los mejores resultados, seguido de Bayes Ingenuo de MLib.

D^*	T^*	N^*	Umbral	Tiempo Apriori	Tiempo Fp-Growth	L^*
5,172	149	44,881	0.2	22seg	24sec	1
			0.1	45seg	26sec	3
			0.05	44seg	24sec	5
			0.02	1min, 35seg	28sec	7
33,716	227	143,334	0.2	07min, 01seg	39seg	1
			0.1	06min, 40seg	43seg	2
			0.05	08min, 05seg	56seg	3
			0.02	43min, 06seg	1min, 41seg	15

Cuadro 7.5: Detalle resultados de conjuntos frecuentes de ENRON. D^* =Número de documentos. N^* =Número de elementos diferentes. L^* =Cantidad máxima de elementos por conjunto

En los resultados de conjuntos frecuentes que se indican en la tabla 7.5, las figuras 7.12 y 7.13. Ambos procedimientos generan la misma cantidad de elementos. Para la primera prueba con 5,172 documentos los tiempos entre Apriori y Fp-Growth son similares. Sin embargo, sobre toda la base de datos Enron con 33,716 documentos existe una diferencia significativa, el menor umbral con 0.2, Apriori tomó 43 minutos mientras que Fp-Growth 1min, 41 seg.

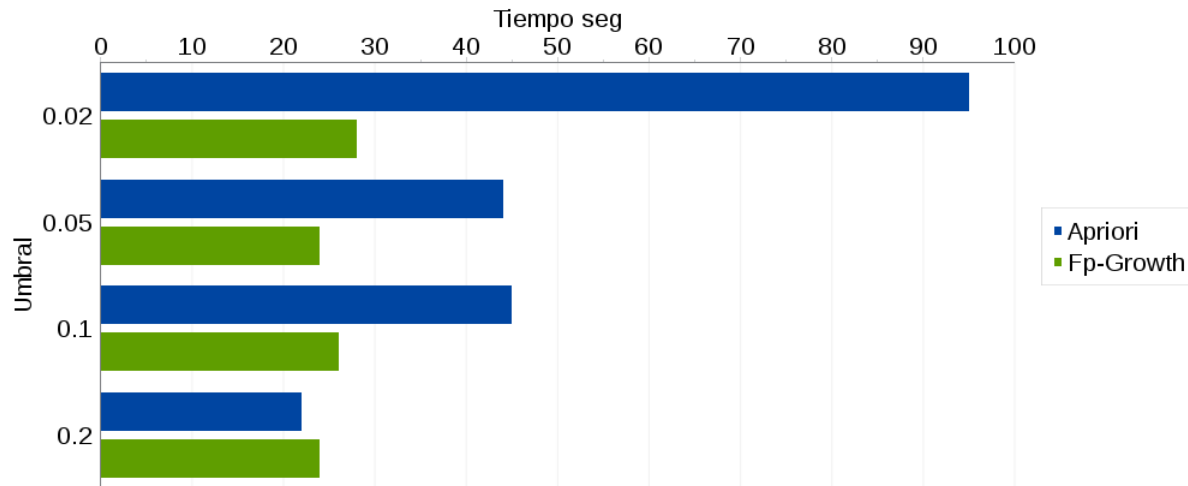


Figura 7.4: Resultados para los tiempos registrados para Apriori con 5,172 documentos.

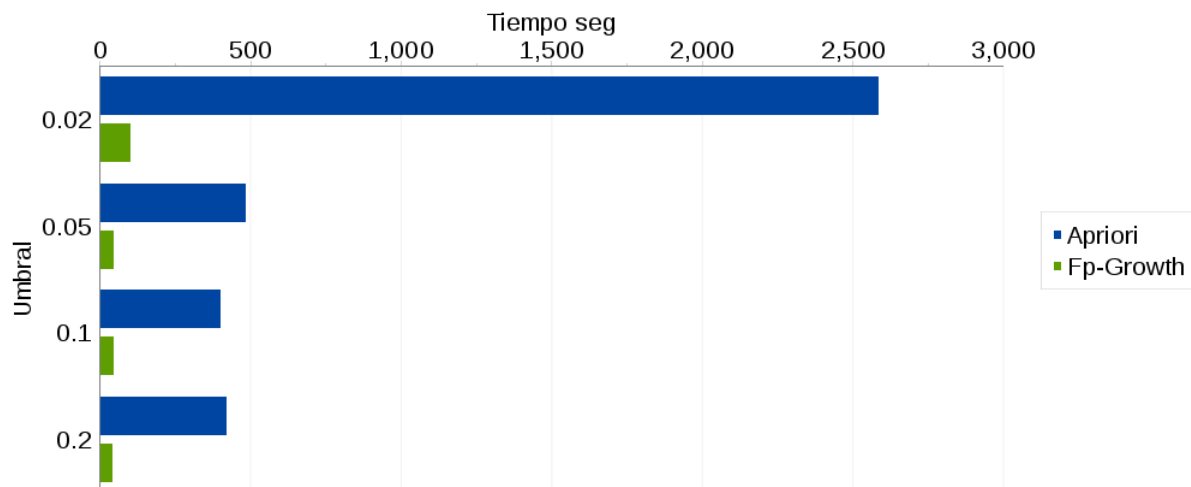


Figura 7.5: Resultados para los tiempos registrados para Apriori con 33,716 documentos.

7.5.2. *RLCP*

Para la base de datos *RLCP*, como se muestra en la tabla 7.6 y figura 7.6 los métodos con la exactitud más alta fueron: Bosques Aleatorios de MLib y *k*NN Join de Hadoop con 0.9999, seguido de Bayes Ingenuo de Spark y MLib con 0.9997. De los valores de MCC registrados los mejores resultados fueron por parte de Bosques Aleatorios de MLib con 0.995 y Spark con 0.994. También es posible ver en la tabla 7.6 y figura 7.7 que el procedimiento que reportó menores tiempos fue Bayes Ingenuo.

Método	Versión	Exactitud	MCC	Tiempo
Bayes Ingenuo	Spark	0.9997	0.972	05min, 11seg
	MLlib	0.9997	0.972	02min, 12seg
	Hadoop	0.9947	.-	.-
Bosques Aleatorios	Spark	0.9974	0.994	30min, 15seg
	MLlib	0.9999	0.995	29min, 01seg
	Hadoop	0.9997	.-	.-
<i>k</i> NN Join	Spark	0.9999	0.990	1d, 15hrs, 01mins, 02seg
	Hadoop	0.9999		
<i>k</i> -Medias	Spark	0.9995	0.894	17min, 09seg
	MLlib	0.9988	0.694	07min, 04seg

Cuadro 7.6: Medidas de calidad de los métodos sobre *RLCP*

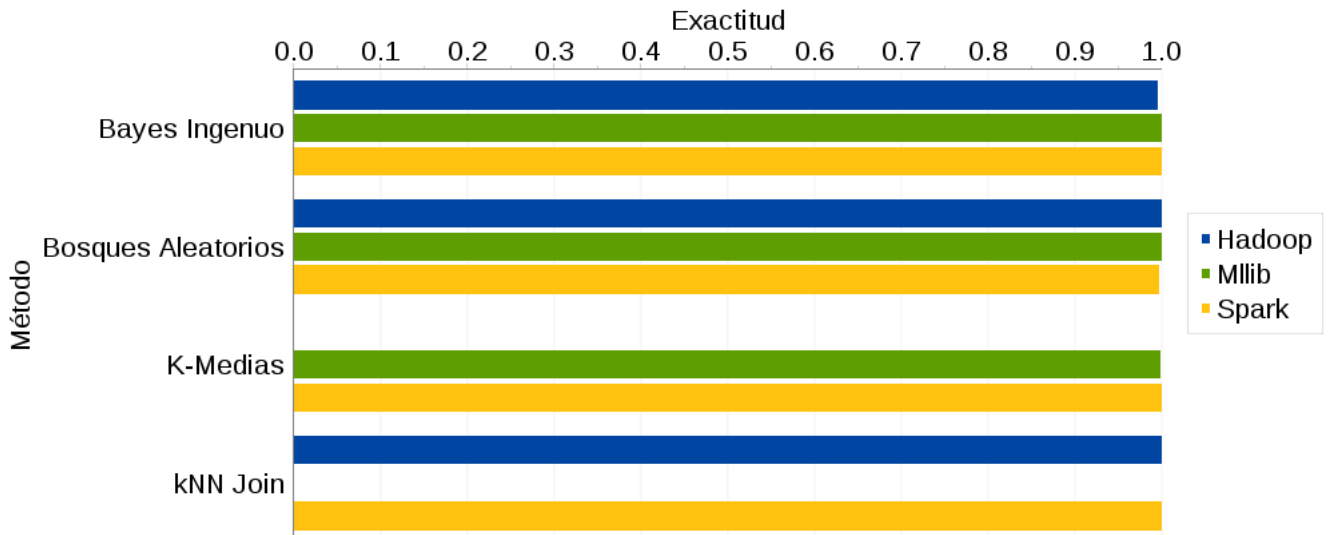


Figura 7.6: Resultados para la medida de desempeño de exactitud de *RLCP*

Una consideración de *RLCP* es que la proporción de las clases es muy diferente con una relación de 200:1. Como se puede observar la medida de Exactitud presenta muy buenos re-

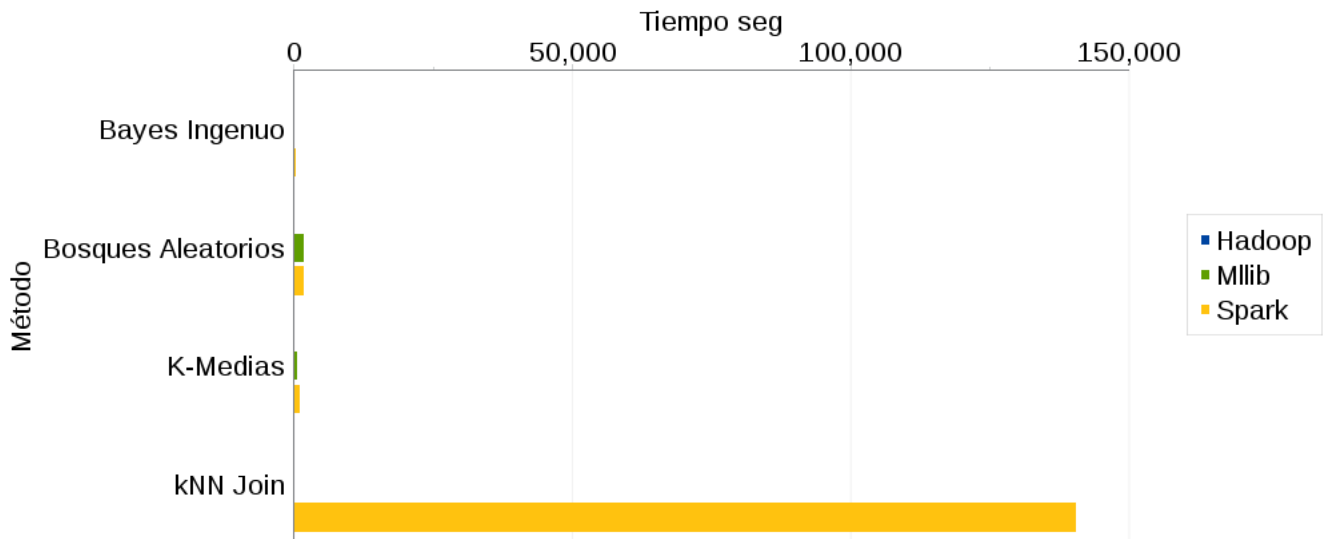


Figura 7.7: Resultados para los tiempos registrados para *RLCP*

sultados en todos los modelos. MCC proporciona una perspectiva más amplia para evaluar el desempeño del modelo. Esto último se presenta en el caso de *k*-Medias que obtuvo resultados similares en Exactitud pero los más bajos en MCC. Esto no es una sorpresa, ya que *k*-Medias no es un algoritmo para clasificación y se esperaba fuera el de menor desempeño.

Método	Versión	FPR	Memoria	Precisión	Valor - F
Bayes Ingenuo	Spark	0.0425	0.9999	0.9998	0.9998
	MLlib	0.0425	0.9999	0.9998	0.9998
	Hadoop	.-	0.9947	0.9999	0.9973
Bosques Aleatorios	Spark	0.0109	0.9992	0.9999	0.9996
	MLlib	0.0036	0.9999	0.9999	0.9999
	Hadoop	.-	0.9999	0.9998	0.9998
<i>k</i> NN Join	Spark	0.0038	0.9999	0.9999	0.9999
	Hadoop	.-	0.9999	0.9999	0.9999
<i>k</i> -Medias	Spark	0.1036	0.9999	0.9996	0.9997
	MLlib	0.3090	0.9999	0.9988	0.9994

Cuadro 7.7: Detalle de medidas de la clase 'No Coincide'

Para los resultados asociados a la clase 'No Coincide' mostrados en la tabla 7.7. La tasa de falsos positivos (FPR) menor fue para Bosques Aleatorios de MLlib seguido de Bosques Aleatorios de Spark. En memoria casi todos presentaron el resultado 0.9999 con excepción de Bayes Ingenuo de Hadoop y Bosques de Spark. En Precisión el valor más alto fue 0.9999 y lo presentaron: Bayes Ingenuo de Hadoop, Bosques Aleatorios de Spark y MLlib, así como *k*NN Join de Hadoop y Spark. En Valor - F el mejor resultado de 0.9999 lo presentaron Bosques

Aleatorios de MLlib así como k NN Join de Hadoop y Spark.

7.5.3. Hepmass

En la tabla 7.8 y la figura 7.8 se observa que los mejores resultados de exactitud los presentó Bosques Aleatorios de Spark con 0.859, seguido de Bosques Aleatorios de MLlib. En MCC los mejores resultados los obtuvo Bosques Aleatorios de Spark seguido de Bosques Aleatorios de MLlib. En la misma tabla y en la figura 7.9 se muestra que los menores tiempos en generar resultados los obtuvo Bayes Ingenuo de Spark y el que más tiempo requirió fue k NN Join.

Hepmass fue la base de datos de mayor tamaño que se procesó con, 10,500,000 millones de instancias. En promedio por iteración se utilizaron 9,500,000 para construir el modelo en la etapa de entrenamiento y 1,000,000 para clasificar. En este caso, es posible notar que hay una diferencia notoria entre el menor tiempo requerido por Bayes Ingenuo y k NN Join.

Método	Versión	Exactitud	MCC	Tiempo
Bayes Ingenuo	Spark	0.800	0.605	47min, 29seg
	Hadoop	0.806	.-	
Bosques Aleatorios	Spark	0.859	0.721	2hrs, 51min, 18sec
	MLlib	0.856	0.713	3hrs, 57min, 17sec
	Hadoop	0.828	.-	
k NN Join	Spark	0.854	0.711	6d, 3hrs, 53min, 52sec
	Hadoop	0.835	.-	
k -Medias	Spark	0.527	0.059	50min, 41seg
	MLlib	0.669	0.351	51min, 03seg

Cuadro 7.8: Medidas de calidad de los métodos evaluados sobre Hepmass

Método	Versión	FPR	Memoria	Precisión	Valor - F
Bayes Ingenuo	Spark	0.139	0.739	0.842	0.787
	Hadoop	.-	0.838	0.759	0.797
Bosques Aleatorios	Spark	0.186	0.904	0.829	0.865
	MLlib	0.180	0.891	0.832	0.861
	Hadoop	.-	0.845	0.819	0.831
k NN Join	Spark	0.184	0.893	0.829	0.860
	Hadoop	.-	0.854	0.823	0.838
k -Medias	Spark	0.505	0.567	0.534	0.544
	MLlib	0.313	0.653	0.694	0.661

Cuadro 7.9: Detalle de medidas de la clase 'Señal'

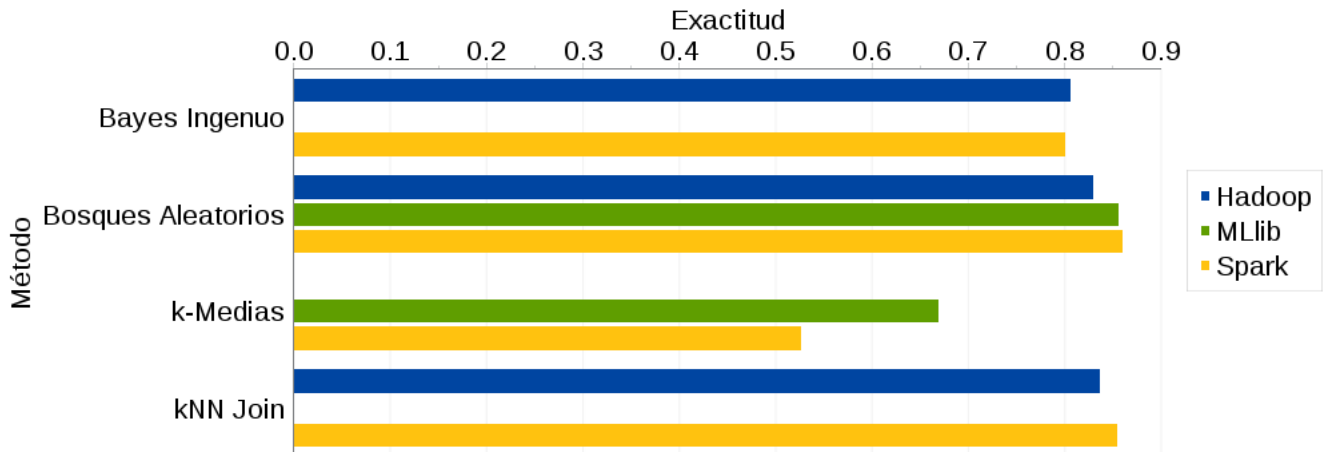


Figura 7.8: Resultados para la medida de desempeño de exactitud de Hepmass

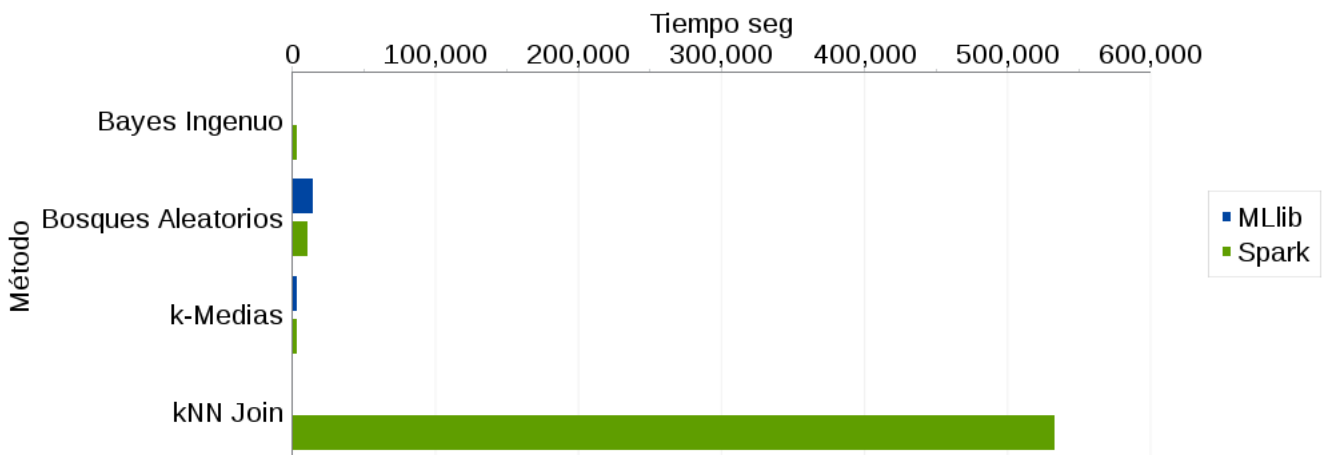


Figura 7.9: Resultados para los tiempos registrados para Hepmass

En la tabla 7.9 se muestran las medidas para la clase 'Señal'. La menor tasa de falsos positivos la obtuvo Bayes Ingenuo de Spark. En Memoria el mejor valor lo presentó Bosques Aleatorios de Spark con 0.904 seguido de k NN Join de Spark. En Precisión los valores más altos lo presentó Bayes Ingenuo de Spark con 0.842 seguido de Bosques Aleatorios de MLlib. En Valor-F el mejor resultado fue Bosques Aleatorio de Spark.

7.5.4. Reuters_4CAT

La última bases de datos utilizada para pruebas fue Reuters.4CAT. En la tabla 7.10 y la figura 7.10 es posible observar que la mejor exactitud la presentó Bosques Aleatorios con 0.908 seguido de k NN Join de Spark con 0.901. En MCC el mejor resultado los obtuvo Bosques Aleatorio de Spark con 0.865. En cuanto a tiempos, en la misma tabla y la figura 7.11 se presenta que el menor lo tuvo Bayes Ingenuo de Spark y el mayor tiempo k NN Spark.

Método	Versión	Exactitud	MCC	Tiempo
Bayes Ingenuo	Spark	0.876	0.835	33min, 29seg
	MLlib	0.855	0.795	58min, 57seg
Bosques Aleatorios	Spark	0.908	0.865	6hrs, 09min, 09seg
	MLlib	0.784	0.691	1hrs, 21min, 39seg
k NN Join	Spark	0.901	0.856	1día, 8hrs, 24min, 44sec

Cuadro 7.10: Medidas de calidad de los métodos evaluados sobre Reuters

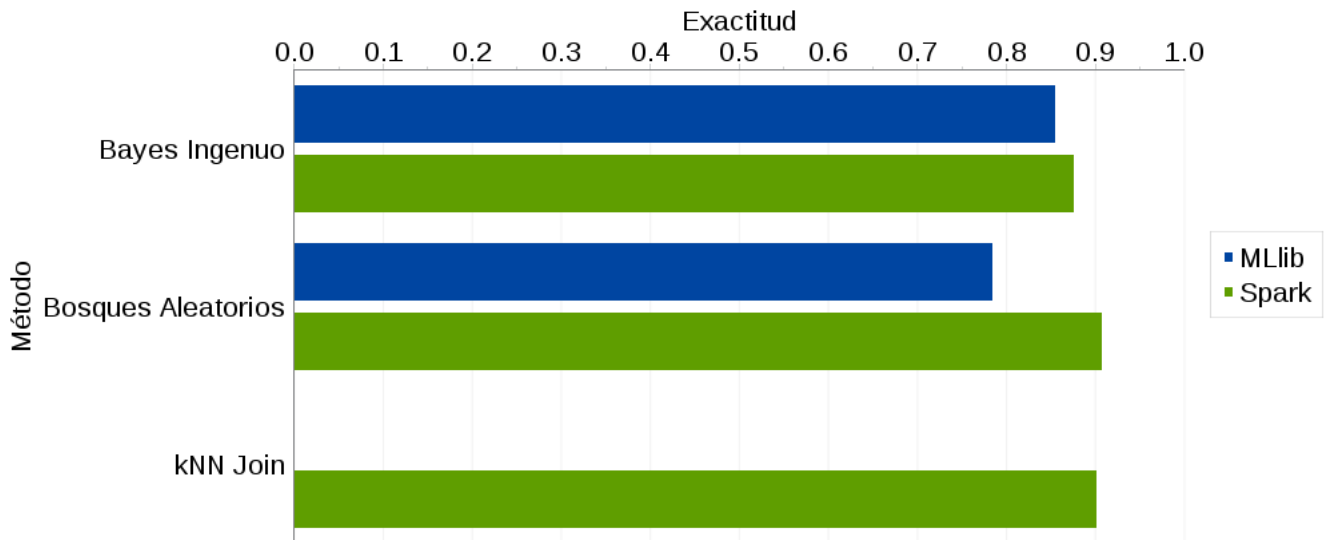


Figura 7.10: Resultados para la medida de desempeño de exactitud de Reuters_4CAT

En la tabla 7.11 se observa que la menor tasa de falsos positivos (FPR) la presentó Bosques Aleatorios de Spark con 0.01615 seguido de k Join con 0.01621. En Memoria el mejor valor

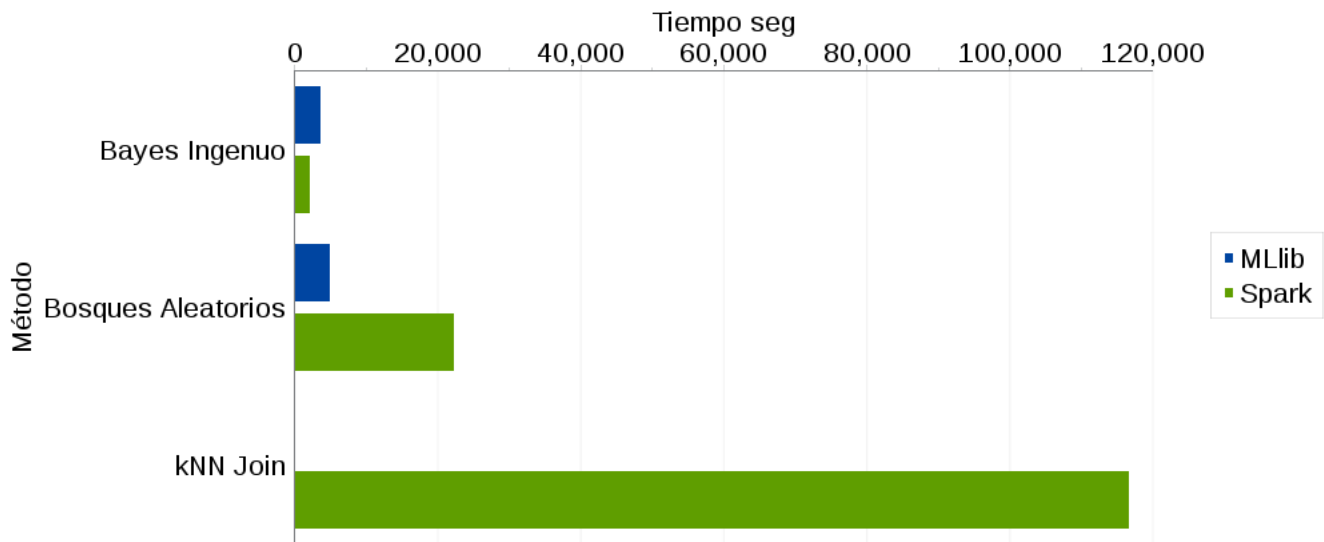


Figura 7.11: Resultados para los tiempos registrados para Reuters_4CAT

Método	Versión	FPR	Memoria	Precisión	Valor - F
Bayes Ingenuo	Spark	0.0266	0.924	0.895	0.909
	MLlib	0.0552	0.846	0.856	0.851
Bosques Aleatorios	Spark	0.03262	0.909	0.917	0.913
	MLlib	0.01615	0.939	0.659	0.775
kNN Join	Spark	0.01621	0.951	0.882	0.915

Cuadro 7.11: Detalle de medidas de la clase 'MCAT'

lo obtuvo k NN Join de Spark con 0.01621. En Precisión el valor más alto lo tuvo Bosques Aleatorios de Spark con 0.917. Mientras que Valor-F el mejor resultado fue para k NN Join con 0.915.

D^*	T^*	D^*	Umbral	Tiempo Apriori	Tiempo Fp-Growth	L^*
273,197	176	183,296	0.2	1hrs, 42mins, 28seg	03mins, 3seg	2
			0.1	5hrs, 56mins, 20seg	03mins, 21seg	3
			0.05	8hrs, 13mins, 16seg	05mins, 23seg	8
			0.02	7hrs, 58mins, 54seg	11mins, 49seg	14
628,333	238	355,704	0.2	2hrs, 57mins, 50seg	06mins, 21seg	1
			0.1	2hrs, 59mins, 51seg	07mins, 37seg	2
			0.05	3hrs, 29mins, 03seg	13mins, 38seg	3
			0.02	hrs, 49mins, 16seg	--	9

Cuadro 7.12: Detalle resultados de conjuntos frecuentes de Reuters_4CAT. D^* =Número de documentos. N^* =Número de elementos diferentes. L^* =Cantidad máxima de elementos por conjunto

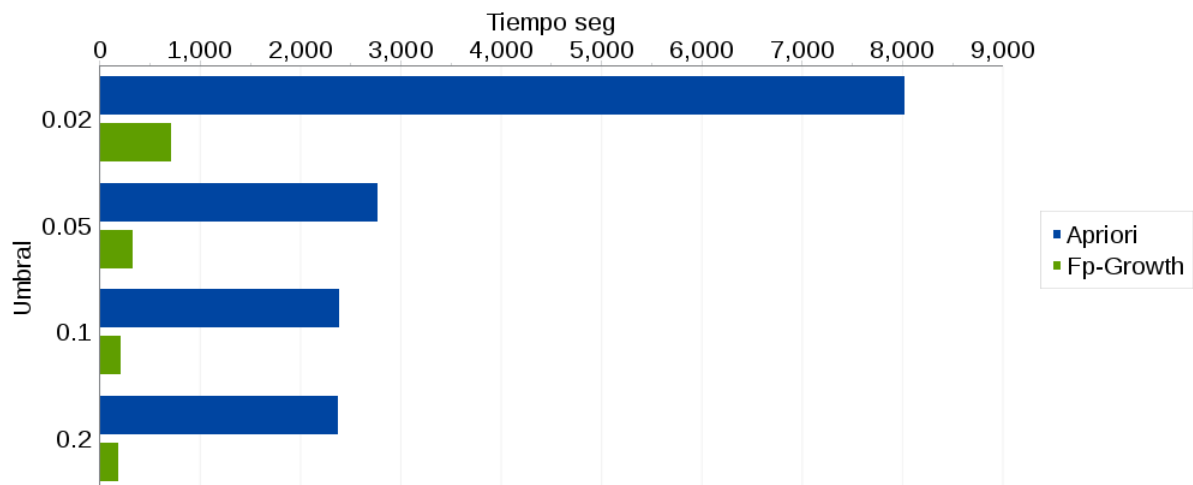


Figura 7.12: Resultados para los tiempos registrados para Apriori con 273,197 documentos.

En tabla 7.12 se muestran los resultados obtenidos por los métodos de conjuntos frecuentes, ambos obtuvieron el mismo conjunto de elementos frecuentes, que se identifica en la columna 'L'. Adicionalmente en la figura 7.12 se observa graficamente que los mejores tiempos para el conjunto con $D=273,197$, fue Fp-Growth. Esto es esperado, ya que es una mejora a Apriori. En este caso la ventaja es que Apriori resulta mucho más sencillo de implementar. En la figura 7.13 se muestran los tiempos para 628,333 documentos. Un problema que se presentó con Fp-Growth de MLib con $D=628,333$ donde se consumía el total de la memoria y no fue posible obtener resultados, se probó con diferente número de particiones.

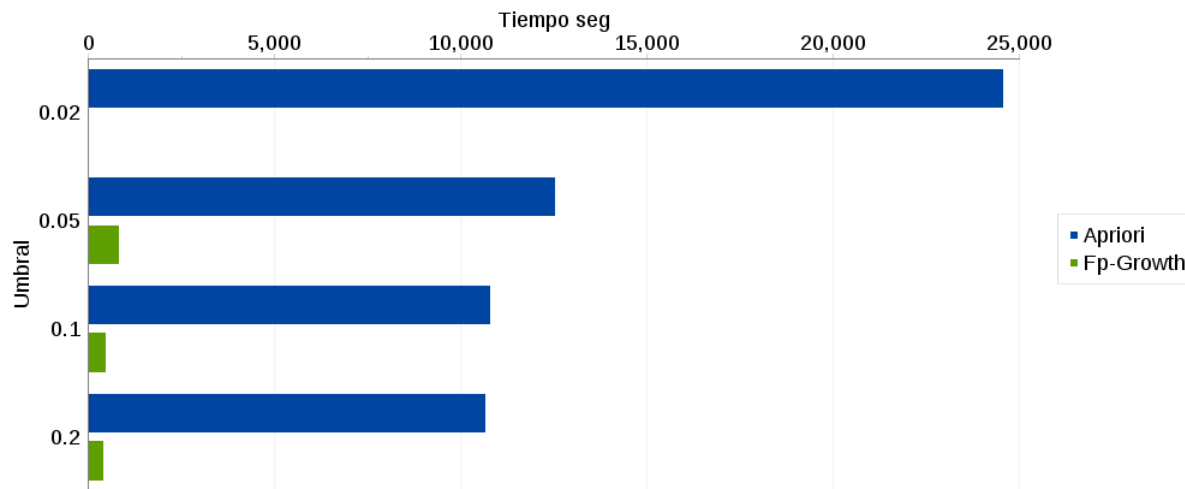


Figura 7.13: Resultados para los tiempos registrados para Apriori con 628,333 documentos.

Configuración que se utilizó en k NN Join donde al tener particiones con menor cantidad de elementos se evitaba el consumo de la memoria dentro de un trabajador. Sin embargo, en este caso a pesar de ajustar este parámetro, el error de consumo de memoria persistió y no se logró obtener los resultados.

7.6. Relación de resultados

Como se puede observar se trata de bases de datos muy diferentes y con características diferentes. Por lo que no es posible comprar los resultados entre ellas. Sin embargo, a continuación se muestran la relación de métodos que presentaron los mejores resultados en la métrica de Exactitud por base de datos. Está fue seleccionada, ya que la presentan todas las versiones. Se descartó MCC, debido a que no se tiene para todos los métodos, y considerando que la relación Exactitud y MCC fue consistente en todas las pruebas. Recordando que la principal motivación de utilizar MCC como métrica adicional de evaluación, fue por *RCLP* que presenta una desproporción en su población.

Método	Primero	Segundo	Tercero
ENRON	Bayes (Spark)	k NN Join (Hadoop)	Bayes (Hadoop)
<i>RCLP</i>	Bosques (MLlib)	Bayes (MLlib,Spark)	Bosques (Spark)
	k NN Join (Hadoop,Spark)	Bosques (Hadoop)	
Hepmass	Bosques (Spark)	Bosques (MLlib)	k NN Join (Spark)
Reuters_4CAT	Boques (Spark)	k NN Join (Spark)	Bayes (Spark)

Cuadro 7.13: Relación de mejores resultados por base de datos

Sobre las base de datos de tipo texto ENRON y Reuters_4CAT, Bayes Ingenuo se encuentra en primero y tercer lugar de resultados. Siendo que en ENRON dos diferentes implementaciones se encuentran entre los resultados más altos. En las bases de datos estructuradas *RCLP* y Hepmass; Bosques Aleatorios y *k*NN Join fueron los que obtuvieron los mejores resultados.

De la relación mostrada en la tabla 7.13 se observa que las versiones implementadas presentan los mejores resultados en tres de las cuatro base de datos. Además que los métodos predominantes en los primeros lugares son Bosques Aleatorios seguido de *k*NN Join y por último Bayes Ingenuo. De los tres tipos de métodos, el que tomó menos tiempo fue: Bayes Ingenuo, seguido de Bosques Aleatorios y por último *k*NN Join.

Capítulo 8

Conclusiones

Este trabajo tuvo por objetivo describir la forma en que se realizan tareas de minería de datos sobre Spark. Motivados por conocer dicha herramienta para ser capaces de llevar a cabo tareas de este tipo sobre grandes volúmenes de datos. Por lo que se seleccionaron cinco algoritmos de minería ampliamente conocidos: Bayes Ingenuo, Bosques Aleatorios, k NN Join, k -Medias y Apriori; los cuáles fueron implementados sobre Spark. Adicionalmente, se utilizaron las versiones de la librería de aprendizaje, MLlib, de Spark, con el propósito de conocer y comparar el desempeño de los métodos que ofrece. La descripción de los métodos implementados se presentaron en el capítulo . En los resultados presentados en la sección sección 7.5 se mostró que las implementaciones realizadas presentan resultados similares las versiones de MLlib y Hadoop, y en algunos casos mejor desempeño.

El proceso fue largo ya que se tuvo que entender los algoritmos así como la forma de operar de la plataforma Spark. Se inició por expresar dichos algoritmos en las operaciones de Spark. Seguido de realizar pruebas en el *cluster* de tres nodos. Los experimentos se realizaron sobre conjuntos de diferentes tamaño. Inicialmente se empezó a trabajar con ENRON, la base de menor tamaño sobre un nodo. Posteriormente al realizar pruebas con *RCLP* y Hepmass fue necesario migrar al *cluster*. Ya que no era posible procesar dichas bases en una sola computadora. Sin embargo, en este escenario se presentaron problemas con el consumo de memoria o un incremento considerado en los tiempos de respuesta. Por lo que hubo que replantear el uso de las operaciones de Spark. Buscando realizar operaciones de forma más eficiente y mejorar los tiempos de respuesta. Con estos ajustes al final fue posible ejecutar todos los métodos sobre las base de datos seleccionadas.

Expresar los algoritmos en Spark es sencillo. El enfoque fue distribuir los datos en bloques (partición) más pequeños, manejables por un nodo, realizar los cálculos de forma local, enviar estos resultados parciales al programa principal para ser sumariados. Siendo este el enfoque de la plataforma Spark, y considerando la naturaleza paralela de los métodos, expresarlos en Spark es sencillo. Una vez que esto se logra, es la plataforma la encargada de la distribución de trabajo y escalamiento. Esto es, no es necesario modificar código al incrementar el conjunto de datos o los nodos del *cluster*. Esto también se observó durante las pruebas. Por ejemplo, por su tamaño la base ENRON si se le aplica el método de Bayes Ingenuo funciona en una computadora como sobre el *cluster*. Así como en las cuatro bases de diferente tamaño a las que se les ejecutaron los mismos métodos.

Durante el desarrollo del proyecto. Una tarea exhaustiva fueron las pruebas, donde en

caso de presentarse algún problema como el consumo de memoria, era necesario realizar los ajustes y volver a iniciarlas para recolectar los resultados requeridos por la validación cruzada. En algunos casos como Hepmass utilizando k NN tomaba días registrar los resultados. Otro aspecto que tomo tiempo, fue el procesamiento previo para las bases de texto.

Para las pruebas realizadas se usaron las implementaciones y las versiones de Spark utilizando el *cluster*. Sobre las bases de datos: ENRON, *RCLP*, Hepmass y Reuters 4CAT. Los resultados para las métricas de desempeño: Exactitud y Tiempo se describen a continuación.

- **Exactitud:** En tres de los cuatros conjuntos de prueba, el método de Bosques Aleatorios presentó los mejores resultados. En dos de estas posiciones se encuentran las implementaciones realizadas. Dentro de las pruebas, lo coloca como un clasificador con la mejor exactitud, independiente al tipo de dato: numérico (para el caso de *RCLP* y Hepmass) y texto (ENRON y Reuters 4CAT). k NN Join fue el método que en segundo lugar obtuvo los mejores resultados, en las dos bases de datos de texto fue consistente y presentó los segundos mejores. El siguiente fue Bayes Ingenuo, obtuvo los resultados más altos para la base de Enron, sin embargo, para hacer uso de la versión de Spark se requiere una matriz de frecuencia de términos, misma que se utilizo con los otros clasificadores, esto impacto de forma negativa en dicha versión, donde se pudo comprobar que Bayes en texto es sensible a la selección de características.
- **Tiempo:** En todos los conjuntos el procedimiento que tomó menos tiempo fue Bayes Ingenuo. Para las bases de datos de tipo texto, la implementación realizada tomó menos tiempo. En el caso de la base numérica *RCLP* fue más rápido presentado resultados en la mitad de tiempo aproximadamente. En segundo lugar, fueron los métodos de k -Medias, en *RCLP* presento menor tiempo MLib, mientras que en Hepmass se requirió tiempos similares en ambas versiones con una diferencia de segundos a favor de la versión implementada. En tercer lugar, se encuentran los procedimientos de Bosques Aleatorios, en las bases de datos de texto la versión de MLib tardó menos tiempo, sin embargo en Hepmass la versión implementada presentó mejores tiempos

En cuanto a los algoritmos de conjuntos frecuentes. Cuando el conjunto era muy pequeño con un umbral de 0.2 (el mayor) se obtuvieron tiempos similares. Sin embargo, en el resto de los casos los mejores tiempos los obtuvo Fp-Growth. Donde se observó que este consume mayor cantidad de memoria, ya que para la última prueba no se pudo obtener resultados ya que consumia los recursos, en ese escenario Apriori si presentó resultados.

Mediante los experimentos fue posible observar que los resultados de los métodos de aprendizaje maquina implementados sobre Spark son similares a los generados por MLib así como las versiones en Hadoop. En algunos casos las versiones implementadas presentan mejores resultados en exactitud o en tiempo. Acercarse a los resultados de MLib y Hadoop es un buen indicador los métodos fueron implementados de forma correcta.

Las bases de datos utilizadas en los experimentos dentro de una categoría de Big Data son relativamente pequeñas. Además el *cluster* estaba conformado por tres nodos. Faltaría

hacer experimentos sobre conjuntos de mayor tamaño así como con una mayor cantidad de nodos.

Se abordó Spark y se mencionó a Hadoop. Sin embargo existen otras herramientas para procesamiento de grandes volúmenes como son: Storm, Flin y Samsa. Que pueden ser utilizados para conocer que tan buenas son para llevar a cabo tareas de aprendizaje. Así como realizar minería en tiempo real.

Los algoritmos implementados son muy utilizados en la literatura, son sencillos y presentan buenos resultados. Una área de oportunidad sería explorar con otros métodos más actuales y explorar su comportamiento con grandes colecciones de datos.

Referencias

- [1] Agrawal, R. and Srikant, R. (1994). Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases, VLDB '94*, pages 487–499, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
- [2] Arthur, D. and Vassilvitskii, S. (2007). K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '07*, pages 1027–1035, Philadelphia, PA, USA. Society for Industrial and Applied Mathematics.
- [3] AWS, A. (2016). Conjuntos de datos p blicos en aws. <https://aws.amazon.com/es/public-datasets/>.
- [4] Breiman, L. (1996). Bagging predictors. *Mach. Learn.*, 24(2):123–140.
- [5] Breiman, L. (2001). Random forests. *Mach. Learn.*, 45(1):5–32.
- [6] Chen, M., Mao, S., and y Yunhao, L. (2014). Big data: A survey. *Mobile Networks and Applications*, 19(2):171–209.
- [7] Dagum, L. and Menon, R. (1998). Openmp: An industry-standard api for shared-memory programming. *IEEE Comput. Sci. Eng.*, 5(1):46–55.
- [8] Dean, J. and Ghemawat, S. (2008). Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113.
- [9] Enron (2015). <https://www.cs.cmu.edu/~./enron/>.
- [10] Fayyad, U., Piatetsky-Shapiro, G., and Smyth, P. (1996). The kdd process for extracting useful knowledge from volumes of data. *Commun. ACM*, 39(11):27–34.
- [11] Fix, E. and Hodges, J. L. (1951). Discriminatory analysis, nonparametric discrimination: Consistency properties. *US Air Force School of Aviation Medicine, Technical Report 4(3):477+*.
- [12] Forum, M. P. (2012). Mpi: A message-passing interface standard version 3.0. Technical report, Knoxville, TN, USA.

-
- [13] Hadoop (2016). <https://hadoop.apache.org/>.
- [14] Hastie, T. J., Tibshirani, R. J., and Friedman, J. H. (2009). *The elements of statistical learning : data mining, inference, and prediction*. Springer series in statistics. Springer, New York. Autres impressions : 2011 (corr.), 2013 (7e corr.).
- [15] HEPMASS (2016). <https://archive.ics.uci.edu/ml/datasets/HEPMASS>.
- [16] IBM, Zikopoulos, P., and Eaton, C. (2011). *Underfording Big Data: Analytics for Enterprise Class Hadoop and Streaming Data*. McGraw-Hill Osborne Media, 1st edition.
- [17] Isard, M., Budiu, M., Yu, Y., Birrell, A., and Fetterly, D. (2007). Dryad: Distributed data-parallel programs from sequential building blocks. *SIGOPS Oper. Syst. Rev.*, 41(3):59–72.
- [18] Jacobs, A. (2009). The pathologies of big data. *Commun. ACM*, 52(8):36–44.
- [19] James Manyika, Michael Chui, B. B. y. J. B. (2011). Big data:the next frontier for innovation, competition, and productivity. mckinsey global institute. http://www.mckinsey.com/insights/business_technology/big_data_the_next_frontier_for_innovation.
- [20] Karau, H., Konwinski, A., Wendell, P., and Zaharia, M. (2015). *Learning Spark: Lightning-Fast Big Data Analytics*. O’Reilly Media, Inc., 1st edition.
- [21] Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2, IJCAI’95*, pages 1137–1143, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
- [22] Laney, D. (2001). 3D data management: Controlling data volume, velocity, and variety. Technical report, META Group.
- [23] Lewis, D. D., Yang, Y., Rose, T. G., and Li, F. (2004). Rcv1: A new benchmark collection for text categorization research. *J. Mach. Learn. Res.*, 5:361–397.
- [24] Li, H., Wang, Y., Zhang, D., Zhang, M., and Chang, E. Y. (2008). Pfp: Parallel fp-growth for query recommendation. In *Proceedings of the 2008 ACM Conference on Recommender Systems, RecSys ’08*, pages 107–114, New York, NY, USA. ACM.
- [25] Lin, M.-Y., Lee, P.-Y., and Hsueh, S.-C. (2012). Apriori-based frequent itemset mining algorithms on mapreduce. In *Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication, ICUIMC ’12*, pages 76:1–76:8, New York, NY, USA. ACM.
- [26] Lloyd, S. (2006). Least squares quantization in pcm. *IEEE Trans. Inf. Theor.*, 28(2):129–137.
-

-
- [27] Loh, W.-Y. and Shih, Y.-S. (1997). Split selection methods for classification trees. *Statistica sinica*, pages 815–840.
- [28] Macías, E. M. (2016). *Minerá de Datos con el modelo MapReduce*. Tesis, Universidad Autónoma Metropolitana.
- [29] Malewicz, G., Austern, M. H., Bik, A. J., Dehnert, J. C., Horn, I., Leiser, N., and Czajkowski, G. (2010). Pregel: A system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, pages 135–146, New York, NY, USA. ACM.
- [30] Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, online edition.
- [31] Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition.
- [32] MLlib (2016). <http://spark.apache.org/mllib/>.
- [33] Odersky, M., Spoon, L., and Venners, B. (2008). *Programming in Scala: A Comprehensive Step-by-step Guide*. Artima Incorporation, USA, 1st edition.
- [34] Quinlan, J. R. (1986). Induction of decision trees. *Mach. Learn.*, 1(1):81–106.
- [35] Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
- [36] Research, Y. (2016). Repositorio de datos de datos de yahoo. <http://webscope.sandbox.yahoo.com/>.
- [37] ReutersRCV1 (2016). <http://trec.nist.gov/data/reuters/reuters.html>.
- [38] Scala (2016). <http://www.scala-lang.org>.
- [39] SciKit-Learn (2016). <http://scikit-learn.org/>.
- [40] Set, R. L. C. P. D. (2016). <https://archive.ics.uci.edu/ml/datasets/Record+Linkage+Comparison+Patterns>.
- [41] Spark (2016). <http://spark.apache.org>.
- [42] Tan, P.-N., Steinbach, M., and Kumar, V. (2005). *Introduction to Data Mining, (First Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.
- [43] UCI (2016). Repositorio de aprendizaje maquina uci. <https://archive.ics.uci.edu/ml/datasets.html>.
-

-
- [44] Wadkar, S., Siddalingaiah, M., and Venner, J. (2014). *Pro Apache Hadoop*. Apress, Berkely, CA, USA, 2nd edition.
- [45] Weka (2016). <http://www.cs.waikato.ac.nz/ml/weka/>.
- [46] Witten, I. H. and Frank, E. (2011). *Data Mining: Practical Machine Learning Tools and Techniques, Tercera Edición (Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
- [47] Wolpert, D. H. (1996). The lack of a priori distinctions between learning algorithms. *Neural Comput.*, 8(7):1341–1390.
- [48] Wu, X., Kumar, V., Ross Quinlan, J., Ghosh, J., Yang, Q., Motoda, H., McLachlan, G. J., Ng, A., Liu, B., Yu, P. S., Zhou, Z.-H., Steinbach, M., Hand, D. J., and Steinberg, D. (2007). Top 10 algorithms in data mining. *Knowl. Inf. Syst.*, 14(1):1–37.
- [49] Yang, Y. and Pedersen, J. O. (1997). A comparative study on feature selection in text categorization. In *Proceedings of the Fourteenth International Conference on Machine Learning*, ICML '97, pages 412–420, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
- [50] Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M. J., Shenker, S., and Stoica, I. (2012). Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, pages 2–2, Berkeley, CA, USA. USENIX Association.
- [51] Zhang, C., Li, F., and Jestes, J. (2012). Efficient parallel knn joins for large data in mapreduce. In *Proceedings of the 15th International Conference on Extending Database Technology*, EDBT '12, pages 38–49, New York, NY, USA. ACM.
-



Casa abierta al tiempo

UNIVERSIDAD AUTÓNOMA METROPOLITANA

ACTA DE EXAMEN DE GRADO

No. 00061

Matrícula: 2143805423

MINERÍA DE DATOS CON SPARK Y SCALA

En la Ciudad de México, se presentaron a las 15:00 horas del día 15 del mes de marzo del año 2017 en la Unidad Iztapalapa de la Universidad Autónoma Metropolitana, los suscritos miembros del jurado:

DR. MARIO GRAFF GUERRERO
 DR. RENE MAC KINNEY ROMERO
 DR. PEDRO LARA VELAZQUEZ

Bajo la Presidencia del primero y con carácter de Secretario el último, se reunieron para proceder al Examen de Grado cuya denominación aparece al margen, para la obtención del grado de:


MAESTRA EN CIENCIAS (CIENCIAS Y TECNOLOGIAS DE LA INFORMACION)

DE: SOFIA ORTIZ VALENZUELA

y de acuerdo con el artículo 78 fracción III del Reglamento de Estudios Superiores de la Universidad Autónoma Metropolitana, los miembros del jurado resolvieron:


Aprobar

Acto continuo, el presidente del jurado comunicó a la interesada el resultado de la evaluación y, en caso aprobatorio, le fue tomada la protesta.



SOFIA ORTIZ VALENZUELA
ALUMNA

REVISÓ




LIC. JULIO CESAR DE LARA ISASSI
DIRECTOR DE SISTEMAS ESCOLARES

DIRECTOR DE LA DIVISIÓN DE CBI



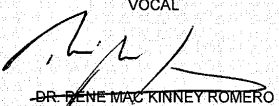
DR. JOSE GILBERTO CORDOBA HERRERA

PRESIDENTE



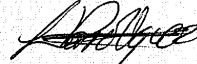
DR. MARIO GRAFF GUERRERO

VOCAL



DR. RENE MAC KINNEY ROMERO

SECRETARIO



DR. PEDRO LARA VELAZQUEZ