Aplicación del algoritmo Kmeans con la plataforma Spark

Carlos Andres Sanchez Alzate
Universidad EAFIT
csanch35@eafit.edu.co

Alejandro Salgado Gómez
Universidad EAFIT
asalgad2@eafit.edu.co

***Palabras clave*—Kmeans, Spark, Big data, TF-IDF**

***Resumen*—Las nuevas técnologías traen consigo cambios en la forma de realizar las labores de creación de algoritmos, pues debido a la cantidad de información que producimos para representar el mundo real en entornos virtuales, se genera la imperiosa necesidad de acelerar el crecimiento de los atributos o cualidades del hardware que producimos. A pesar de que las tecnologías de paralelización existen ya hace algunos años, con este crecimiento abrumador de la información que producimos, nos vemos en la obligación de migrar a un razonamiento en paralelo respecto a la producción de algoritmos. En este articulo veremos como la programación paralela efecta la solución del problema respecto al timpo de respuesta que esta nos brinda, tomando como base un problema cotidiano como lo es el procesamiento del lenguaje natural y la minería de datos para la clasificación de libros.**

1. Introducción

Muchos de los datos que encontramos en la web, están dispuestos en forma de texto y generalmente están allí con el proposito de informar o expresar ideas y difundir información de una forma más efectiva. Entender o descifrar el significado de las símobolos plasmados en textos es una caracteristica de los humanos, pero con el auge de la producción masiva de información que conlleva al crecimiento de la producción textual por medios digitales, carecemos de la habilidad de procesar cientos o miles de ellos para llevar a cabo tareas más complejas como la clasificación en un tiempo razonable.

Para las máquinas también se dificulta esta tarea cuando la cantidad de información que se procesa es muy grande, puesto que estás también tienen recursos limitados y para aprovechar al máximo el uso de estos recursos debemos conocer muy bien la arquitectura que poseen, pues estás actualmente cuentan con la capacidad de procesar información en paralelo, caracteristica que nos permite disminuir el tiempo de computo y completar de forma más rápida las tareas que pretendemos realizar.

En este ariticulo se explicará y analizará una metodología para la optimización de un algoritmo de clasificación de textos y como elaboramos un modelo que nos permita representar de forma adecuada los texto que se desean clasificar. El objetivo de este trabajo es utilizar técnicas de análisis sobre grandes volumenes de información para lograr el objetivo planteado anteriormente.

2. Descripción del problema

En la actualidad el NPL (Natural Processing Language), es una área de estudio que ha cobrado mucha importancia para el procesamiento y entendimiento del leguaje natural realizado de forma automática, esto ha permitido dotar a las máquinas de la capacidad de ejecutar tareas complejas que serían dificiles o imposibles de realizar por el ser humano.

La clasificación en la minería de datos es uno los problemas usados para la exploración de soluciones en divesos problemas de manipualción de textos, estas técnicas nos dotan de la capacidad de categorizar y detectar similaridades entre grupos de datos para organizarlos bajo criterios de distancias entre cada uno de sus elementos, y cuyos componentes se relacionan entre si por su cercanía, de modo que al agruparlos bajo este criterio podemos generar subgrupos donde sus elementos aducen caracteristicas similares, asi se puede identificar con mayor facilidad peculiaridades generales de los datos que analizamos.

2.1. TF-IDF

Para la extracción de características de los documentos, en este trabajo se utilizará la implementación del algoritmo de TF-IDF para representar los textos numéricamente a partir de palabras que los conforman. TF-IDF tiene diversas aplicaciones, entre ellas están el reconocimiento de patrones y la clasificación de documentos, bajo este último enfoque, es un método que es usado para representar documentos ignorando su gramática y el orden de las palabras, este algoritmo logra obtener información sobre la importancia de los terminos que componen un documento.

TF-IDF nos permite generar una reprecentación numérica a partir de textos con las cuales podemos utilizar como entrada a diversos algoritmos de clasificación, por ejemplo kmeans. Dichas representaciones son generadas a

partir de una funcion hash, la cual es usada para obtener una primera representación del documento y posteriormente es utilizada para priorizar los terminos del texto.

2.2. K-Means

En términos generales, clustering tiene como objetivo dividir un conjunto determinado de objetos en grupos, de modo que los objetos en el mismo grupo sean similares y los objetos en diferentes grupos sean diferentes entre sí. El término objetos se usa para referirse a entidades, observaciones o puntos de datos en un conjunto de datos. El término clúster, aunque no tiene una definición precisa, describe un grupo de elementos.

El clustering con K-means es un algoritmo de clasificación y un técnica de entrenamiento no supervisada, y es el mejor representante del clustering duro (No difuso). Por clústeres duro nos referimos a los algoritmos de clustering que intentan dividir los datos disponibles en algunos subconjuntos para que cada punto de datos solo puede pertenecer a un subconjunto.

En particular, K-means tiene como objetivo dividir un conjunto de n puntos de datos x1, x2,... , xn en K clusters C1, C2,. . . , CK tal que la función de costo J se minimiza:

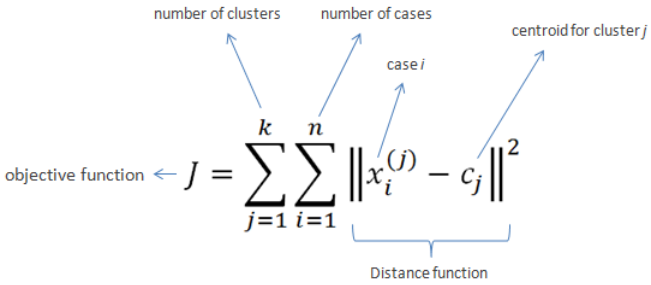

Figura 1. Función objetivo del algoritmo Kmeans
[1]

En el conjunto de observaciones x1, x2, ..., xn, cada observación es un vector real con d dimensiones, el algoritmo construye k subconjuntos de las observaciones con el fin de minimizar la suma de cuadrados dentro de cada grupo.

Para la implementación de k means se pueden usar diferentes criterios de medida para calcular las distancias de los puntos a los centroides, en particular en la ecuación 1, se hizo uso de la distancia euclideana y está será la que seguiremos utilizando en este trabajo.

Abajo detallaremos un algoritmo básico de K-means, el cual consta de 5 pasos.

1. Seleccione aleatoriamente K puntos de datos del conjunto de datos de entrada y asignelos como centroides de los clústeres c1, c2,..., cK.
2. Calculamos la distancia entre los centroides y cada uno de los datos.
3. Calculamos la posición de los centroides de modo que estos se ubiquen aproximadamente en el centro del cluster.
4. Repetimos del el paso 2 y 4 hasta cumplir con los criterios de parada que se definan.

Este algortimo de clustering puede ayudarnos a revelar la estructura subyacente de los datos y, por este motivo,se ha convertido en una de las técnicas más utilizadas para el análisis exploratorio de datos en diferentes campos de la ciencia y la ingeniería.

3. Análisis y diseño

Para la implementación del algoritmo se tuvieron en cuenta varios factores, el primero de ellos fue la forma de represantar los textos de tal manera que se les pueda calcular una medida de distancia. Para lograr este objetivo es necesario representarlos de forma numérica, por ello se hacía necesario el uso de un algoritmo que nos permitiera traducir los textos en vecotres de carácteristicas sobre los cuales se pudieran efectuar operaciones matemáticas. Después de analizar varias alternativas, se decidió utilizar el algoritmo TF-IDF.

TF-IDF es un algoritmo compuesto por 4 etapas, la primera de ellas es la representación de los documentos en una primera version numérica, para esto se utiliza una función hash.

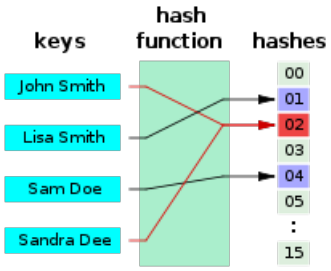

Figura 2. Función hash
[2]

Dicha función toma las palabras de cada uno de los textos y devuelve un identificador numérico que la representa. Luego está representación es usada para calcular la frecuencia de las palabras en el texto, esta etapa es llamada TF (Term frequency). Es necesario tener en cuenta que debido a la manera en la que opera una función hash pueden presentarce colisiones, lo que siginifica que dos palabras distintas puedan ser asignadas al mismo número, para disminuir este problema se puede aumentar el rango de la función, en este caso se utilizó 2^{20}, el cual es el

estandar de la libreria de machine learning de Spark.

Posteriormente se calcula la importancia de cada una de las palabras basados en el siguiente modelo

$$IDF(t, D) = log\left(\frac{|D| + 1}{DF(t, D) + 1}\right) \quad (1)$$

Donde D es el conjunto de documentos a ser analizados, t representa el término al que se le desea calcular su importancia, $|D|$ representa la cardinalidad del conjunto D, es decir la cantidad de documentos, y $DF(t, D)$ representa la cantidad de documentos en la que el término en cuestión aperece.

Para entender mejor la forma de esta función es necesario recordar ciertos aspectos, el primero de ellos es que una división no esta definida si su denominador es 0, por esta razón se debe agregar un 1 al cálculo de la función $DF(t, D)$ para asegurar que la división siempre este definida. En segundo lugar se debe recordar que el logaritmo se anula en $x = 0$, tal como se muestra en la Figura 3, lo que explica la adición de una unidad al cálculo del numerador para evitar que la función $IDF(t, D)$ resulte en un valor nulo. Finalmente es necesario notar que la desigualdad $|D| >= DF(t, D)$ siempre es verdad, lo que implica que la división tiene el dominio $[1, \infty)$, tal como se muestra en la Figura 4, lo que significa que la expresión trabajada siempre resultará en valores positivos, los cuales son acotados por la función logaritmo.

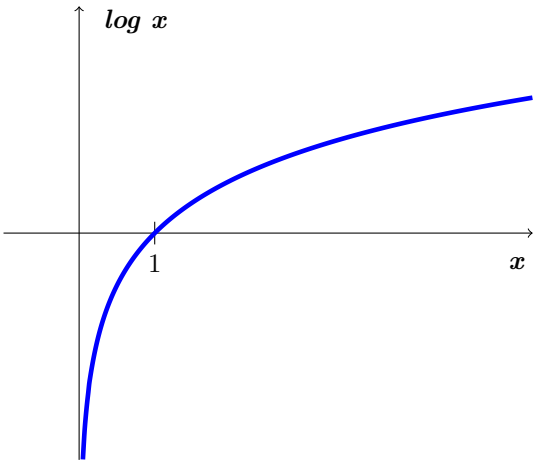

Figura 3. Función logaritmo

Una vez entendido esto se puede observar como la función $IDF(t, D)$ provee información sobre la importancia de un término, esto se debe a que tiene en cuenta la relación entre la cantidad de documentos que contienen el término en cuestion con la cantidad de documentos totales, lo que implica que los términos que son bastante repetidos tendrán una importancia baja, resultando en una influencia baja en la decisián de clasificación, mientras que los términos más unicos seran clasificados con un valor más alto, implicando que tendrán una importancia mayor en la etapa de clasificación.

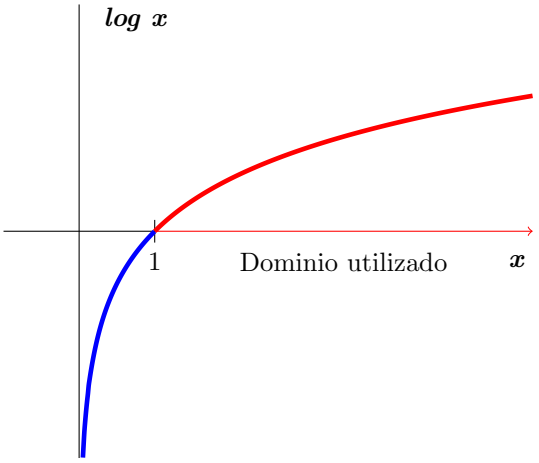

Figura 4. Parte usada de la función logaritmo

Finalmente la clasificación total de los terminos es calculada como la multiplicación entre el factor TF y IDF para asociar tanto la cantidad de veces que un término aparece en el texto, como su importancia general, obteniendo así la representación numérica final que sera usada como entrada para el algoritmo Kmeans.

Ahora, en cuanto al rendimiento de esta nueva implementación, se espera que los tiempos necesarios para completar la ejecución sean mayores que la versión realizada en el proyecto pasado, esto por varias razones, la primera es la plataforma de ejecución, debido a que en la practica de HPC se utilizaba el hardware directamente desde la aplicación, el rendimiento podia subir más de lo que se podria lograr con las máquinas virtuales del cluster Hadoop, incluso teniendo en cuenta que en esta última plataforma se hacia uso de una arquitectura distribuida. En segundo lugar el el lenguaje de programación, en la práctica pasada se utilizo C++ con Cuda para implementar el algoritmo, mientras que en éste proyecto se utilizó Python, lo que agrega capas de abstracción, las cuales facilitan la programación, pero aumentan los tiempos de ejecución. Finalmente tal como se hablo en la practica pasada, se encontró que una implementación en GPU resulta en menores tiempos de ejecución gracias a su arquitectura altamente paralela y su rapida creación de procesos.

Sin embargo, una implementación como la que se presenta en esta práctica no siempre tendrá un rendimiento menor a una diseñada para ejecutar en un ambiente HPC, de hecho es necesario que estos algoritmos puedan ser implementados en arquitecturas diseñadas para el tratamiento de grandes volumenes de datos, tal como un cluster Hadoop, debido a que, apesar de su velocidad, una infraestructura

HPC perdería gran parte de su rendimiento en casos en los que la cantidad de datos sea grande, mientras que arquitecturas como la utilizada en este proyecto mostrarian todo su potencial logrando tiempos de ejecución y costos mucho menores. Por esta razón es necesario hacer la aclaración de que se espera un rendimiento menor que en la practica de HPC, pero esto se debe al tamaño de los datos utilizados.

4. Implementación

Teniendo en cuenta que se trata un problema en el cúal puede estar involucrado el procesamiento de granades volumenes de información, se hace uso de herramientas diseñadas especialmente para este proposito, como lo son Apache Spark y Hdfs del ecosistema Hadoop, quienes son utilizados respectivamente para el procesamiento de la información y su almacenamiento.

Para la clasifiación de los documentos, se hace uso de las librerías que provee Spark, de las cuales algunas de las funciones implementadas están diseñadas para éste propósito. Apache Spark posee funciones para la transformación de los documentos en representaciones numéricas o cuantitativas, en este trabajo se hace uso de TF e IDF, las cuales realizan el procesaminto de los archivos de textos como se explicó en la sección de análisis y diseño. Luego de la transformación de los documentos se procede a utilizar kmeans, función qué se encuentra implementada en la librería Mlib de Apache Spark.

Para la transformación del documento en una representación numérica correcta en la clasificación con Kmeans, se requiere utilizar dos funciones que Spark provee, las cuales son TF e IDF.

La función TF calcula la frecuencia de las palabras dentro del documento. A ésta se le debe pasar como parametro el documento que queremos transformar y lo hacemos de la siguente manera.

```
tf = hashingTF.transform(documents)
```

Luego procedemos a calcular los valores de la frecuencia inversa de las palabras de los documentos, lo que nos permite conocer la importancia de las palabras en estos. A esta función le pasamos como parámetro la variable TF, quien fue calculada con anterioridad.

```
idf = IDF(minDocFreq=5).fit(tf)
```

Para términar con la transformación debemos multiplicar TF e IDF, y dicho valor nos dará los valores finales de la importancia de cada una de las palabras dentro de los documentos.

Mlib nos provee el algoritmo Kmeans, quien nos permite llevar a cabo el último paso para la clasificación de documentos, pues este nos permite separarlos y asignarlos a un conjunto o categoría.

La función Kmeans tiene 3 parametros que son necesarios para ejecutar la clasificación, estos son el K o número de clusters que deseamos, el número máximo de iteraciones para la convergencia del algoritmo y el datasets con valores numéricos, este último es dónde cobra sentido la realización de los indices invertidos y la frecuencia de las palabras, pues kmeans no permite la clasificación de variables categóricas.

```
clusters = KMeans.train(tfidf, k, \
                        maxIterations=maxIter)
```

El algoritmo al ser ejecutado por Spark, los resultado siguen conservando la forma en la que son almacenados en memoria por RDD, por lo que debemos hacer un colect sobre los nombres de los grupos y sobre los resultados de la clasificación para agruparlos y conocer que documento pertenece a que cluster.

```
clusterid = clusters.predict(tfidf) \
                    .collect()

names = names.collect()
```

5. Resultados

En esta sección se muestran los resultados de ejecutar el algorimo Kmeans en la plataforma Spark del datacenter academico y posteriormente se comparan los resultados con la implementación creada en el proyecto pasado. Se realizaron 6 tipos de pruebas, todas con el mismo subconjuntos del dataset Gutenberg, el cual fue tomado de la carpeta compartida del datacenter academico, las pruebas realizadas fueron:

- Ejecución solo en el master del cluster, a estas pruebas se les denomina local.
- Ejecución distribuida con 2, 3 y 4 procesadores, a estas pruebas se les denomina distribuido 2, 3 y 4 respectivamente.
- Ejecucion de la version serial de la practica pasada, a esta prueba se le denomina serial.
- Ejecucion de la version paralela de la practica pasada, a esta prueba se le denomina paralela.

Al ejecutar las primeras pruebas sobre la arquitectura Spark se encontro como a medida que se utilizaban más procesadores y era necesaria mas comunicación entre los diferentes procesos que componen la ejecución del algoritmo, el rendimiento se deteriora cada vez más siguiendo un

comportamiento casi lineal, tal como se puede observar en la Figura 5.

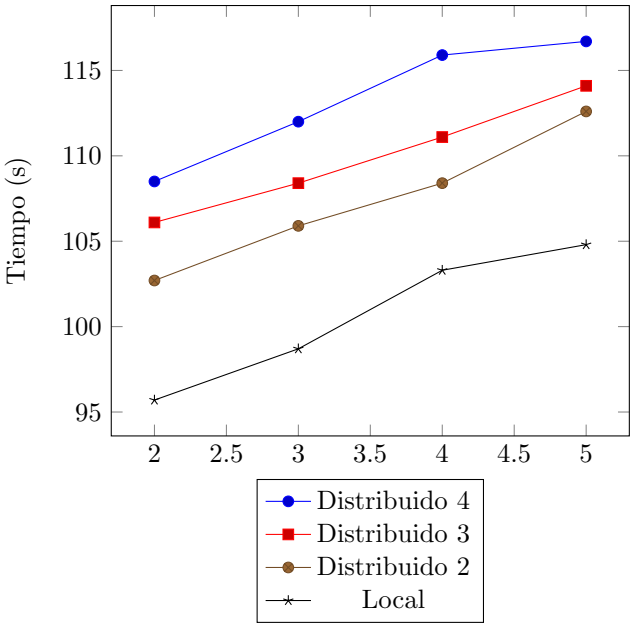

Figura 5. Resultados Spark

Finalmente, tal como se esperaba, al comparar los rendimientos de la implementación en Spark y la de HPC se puede observar una notoria diferencia en los tiempos de ejecución, tal como se muestra en la figura 6, donde se puede observar como los tiempos de ejecución de la plataforma Spark se incrementan más de lo que lo hacen las implementaciones serial y paralela del proyecto pasado.

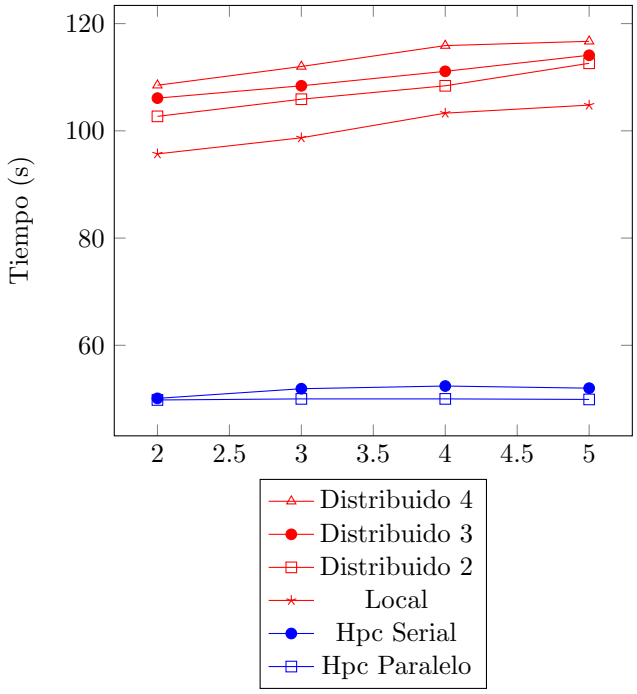

Figura 6. Resultados combinados

Se considera conveniente volver a recalcar que las ejecuciónes de la plataforma Spark pierden rendimiento a medida que se aumentan procesadores debido a que el tamaño del dataset utilizado en la ejecución del algoritmo es pequeño, lo que provoca que el tiempo utilizado en la comunicación entre los procesadores se vuelva más notaria y termine afectando de manera negativa el rendimiento.

6. Conclusiones

En en el tratamiento de grandes volumenes de información, y con la variabilidad de los datos o el almacenamiento de datos no estructurados, las arquitecturas de hardware y las soluciones de software que brindan los conceptos de las tecnologías de big data a través de los frameworks como Hadoop o Spark, hacen posible el procesamiento y la manipulación de la información, orientada al análisis para la adquisión de conocimiento a partir de estos.

En este estudio encontramos que los algortimos programados con tecnologías big data, no tenian un mejor rendimiento que los algoritmos realizados en la práctica anterior, en la que el rendimiento era un punto clave y por lo que se plantearon estrategias basadas en el conocimiento de la arquitectura del hardware para sacar el mayor provecho de estos recursos y lograr un buen rendimiento a través del procesamiento en GPU. Por el contrario en la elaboración de éste trabajo se analizó más el como tratar los datos desde el punto de vista del almacenamiento y como almacenarlos para procesarlos

a partir del uso de herramientas que proveen los frameworks.

En la elaboración de la práctica no se hizo necesario un analisis muy profundo del uso de la arquitectura del hardware y como utilizar esta para optimizar el tiempo de ejecución de los algoritmos, esto devido a que las herramientas de software que proveen los entornos de desarrollo para big data abstraen esto agregando capaz adicionales para que sea hasta cierto punto transparente para el usuario. El caso de esta implementación se limitó al uso de librerías que provee Apache Spark para hacer minería de textos.

Los sistemas que usan tecnologías de big data están surgiendo rápidamente como uno de los sistemas más críticos en el entorno de TI de las organizaciones, pero con una cantidad masiva de datos, surgen problemas de rendimiento. Si los sistemas big data no se pueden usar para hacer pronósticos para la toma de decisiones comerciales críticas, o para proporcionar información sobre valores comerciales ocultos bajo grandes cantidades de datos en tiempos de computo razonables, estos sistemas pierden su relevancia.

Por lo anterior es importante pensar en el rendimiento a la hora de desarrollar para está técnología y deja en nuestras manos la labor de investigar para seguir profundizando y mejorando las soluciones que generamos.

Referencias

[1] Introduction to k-means clustering. http://nithinnaithottu.blogspot.com.co/. Consultado el 17 de Noviembre de 2017.

[2] Hash function. https://en.wikipedia.org/wiki/Hash_function. Consultado el 17 de Noviembre de 2017.