Tópicos Especiales en Telemática: Proyecto 4 – Clústering de Documentos Usando Spark

Pablo Cano
pcanogo@eafit.edu.co
201427501010
EAFIT

PALABRAS CLAVE: k-means, Big Data, Clúster, Centroides, Spark, Pyspark, HDFS, RDD, Dataframe.

ABSTRACT: En este documento se demostrará el análisis e implementación de un algoritmo de k-means hecho en pyspark para la creación de clústeres de documentos de texto. La implementación fue hecha en Python y cuanta con la medición de palabras usando el método TF-IDF.

1. INTRODUCCIÓN: La clasificación del objeto muy común en los servicios virtuales hoy en día. El caso más común es como Netflix es capaz de recomendar películas o series de acuerdo con el contenido que has visto. Esto es gracias a los algoritmos de aprendizaje que están detrás de las escenas que permiten clasificar lo que el usuario es consumiendo. Pero la clasificación no solo se usa para el contenido visual, también es aplicada en textos como en este caso.

La clasificación de textos es una tarea que requiere de un alto rendimiento de una máquina. Las operaciones que tienen que cumplir son extensas y se necesitan de varias iteraciones. El número de recursos necesitados para llevar a cabo dicha tarea aumentan significativamente cuando se trata de datasets extensas como la librería virtual de Amazon Books.

En Muchos casos la única forma de poder cumplir con la expectativa de los usuarios es necesario traer la ayuda de High Performance Computers y un algoritmo diseñado para correr paralelamente.

2. MARCO TEÓRICO: En base, el problema era desarrollar una implementación de k-mean con pyspark y la librería de sparks MLib para la calcificación de textos. Esta debe incluir el proceso ETL de los datos y la medida de pesos de palabra con TF-IDF.

2.1. Representación de textos

Antes que todo, los documentos primero deben pasar por un proceso de normalización para que se puedan hacer las operaciones requeridas en el algoritmo. Este es un proceso que extrae la información que tiene valor en el texto y representarla en estructura de vector. Lo más importante aquí es extraer las palabras que verdaderamente dan información decisiva del documento. Es se tiene que llevar a cabo con una implementación de una matriz Term Frequency – Inverse Document Frequency (tf-idf).

2.2. K-means

Ya cuando se tiene la matriz de tf-idf se puede continuar con la implementación de k-mean. Este algoritmo no es uno complicado. La teoría es sencilla, inicializar unos Centroides que deben ser igual la cantidad de agrupaciones que se quieren encontrar. Luego se calculan las distancias de cada documento con cada Centroide y se asigna al clúster que corresponde al Centroide más cercano. Una vez estén asignado todos los documentos se calculan nuevos Centroides que son igual a la mediana de todos sus documentos asignados. Este ciclo se repite hasta que haya convergencia con todos los Centroides.

3. Análisis y Diseño

3.1. ETL

Para este proyecto es importante implementar el proceso ETL para los datos. ETL significa "Extract, Transform and Load". Asi que esto implica extraer los datos, transformarlos en una forma entendibles y loa cargar la data en el sistema deseado, en este caso el sistema será kmeans.

3.2. Extract

La extracción de los datos es fácil gracias a las herramientas que brinda Spark. La extracción de datos se hacer por medio de RDD. Esto sea hace creando un contexto de Spark y especificándole que lea el directorio de donde se encuentran los archivos a analizar.

3.3. Transform

En la transformación lo primero que hacemos es pasar el RDD creado con los documentos y transformarlos en un dataframe. Esto permite que los datos sean mucho más legibles a la hora de leer los resultados.

Después siguen la serie de transformación que son necesarios para un k-means efectivo. Esto incluye tokenizar las palabras para que sean fácil de manipular y hacer cada operación sobre palabra. Después quitar los stopwords, estos son las palabras que no traen ningún valor informativo sobre el texto como "a", "por", etc. Luego se hace un conteo de la frecuencia de palabras y por último se crea un vector con las palabras de cada texto que más "peso" tienen.

3.4 Load

Esto paso ya lo implica cargar las palabras en el algoritmo de k-means de Spark. Este ya retornara la predicción de cual clúster pertenece cada texto.

4. IMPLENTACION:

Para esta implementación se usó Python con la librería de pysparks para acceder a los recursos de Spark.

Crear RDD: Haca se utiliza el método principal de extracción de datos de sparks. Para hacer esto sea crea un contexto Spark y luego creamos una variable que crea un diccionario del path del documento y el contenido de los documentos.

```
sc = SparkContext()
spark = SparkSession(sc)

documents =
sc.wholeTextFiles("hdfs:///distributed/
file/directory")
```

Transformar RDD en Dataframe: Cuando tenemos el RDD con el contenido de los documentos hay que crear un dataframe para que la información este representada de una manera legible. Para esto creamos primero un schema que va ser el cuerpo del dataframe y ya con eso más el rdd creamos el dataframe.

Para el resto de transformación se tiene que definir la transformación primero y despes se aplica la acción de transformar.

```
schema =  StructType([StructField
("path" , StringType(), True) ,
StructField("text" , StringType(),
True)])
docDataFrame =
spark.createDataFrame(documents,schema)
```

Tokenizar las palabras: Se crea una lista del documento con cada palabra que contiene. Este recibe la columna del contenido del texto y crea un dataframe con las columnas del dataframe anterior más una columna extra con la lista de palabras de cada texto.

```
remover =
StopWordsRemover(inputCol="terms",
outputCol="filtered terms")

wordsData =
tokenizer.transform(docDataFrame)
```

Quitar los stop words: Aquí quitamos as palabras que se repiten mucho pero no dicen nada sobre el texto.

```
remover =
StopWordsRemover(inputCol="terms",
outputCol="filtered terms")


filteredData =
remover.transform(wordsData)
```

Hash TF: Aquí creamos una lista de frecuencia de palabras para que con esto podamos hacer la matriz de tf-idf. Hay que notar que en este proceso no se hizo stemming de las palabras porque la transformación de hashTF hace esto por nosotros. Aquí también designamos cuántos "features" queremos calcular, mientras más features más exacto es la predicción.

```
hashingTF =
HashingTF(inputCol="filtered terms",
outputCol="term frequency",
numFeatures=20)
```

IDF: Aquí calculamos la frecuencia inversa de los términos. Para esto se usa la fórmula:

$$tfidf(d,t) = tf(d,t) \times log(\frac{|D|}{df(t)})$$

Hay que notar que para hacer esta transformación también debemos cargar la data en un modelo.

```
idf = IDF(inputCol="term frequency",
outputCol="features", minDocFreq=1)
```

```
idfModel = idf.fit(featurizedData)

rescaledData =
idfModel.transform(featurizedData)
```

TF-IDF: Después de hacer todas las transformaciones podemos usar la implementación de k-means de MLib.

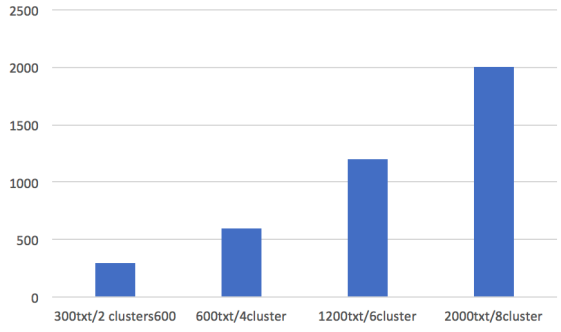
Para esto también se debe cargar la data sobre un modelo primero. Esta última acción crea un dataframe con toda la información creada por cada transformación y agrega una columna donde está el resultado de la predicción que es un numero dependiendo del número de clústers.

```
kmeans = KMeans(k=4)

kmeansModel = kmeans.fit(rescaledData)

predictionData =
kmeansModel.transform(rescaledData)
```

5. ANÁLISIS DE RESULTADOS: Para las pruebas seriales del algoritmo se ejecutó el programa 4 veces. La primera vez con solo 300 textos y 2 clústeres. La segunda con 600 textos y 4 clústeres. La tercera vez 1200 y 8 clústeres. La última vez 2000 textos y 12 clústeres. Abajo podemos observar los resultados:



Aquí podemos comparar los tiempos obtenidos con la implementación hecha en el módulo 3.

No. Archivos	Paralelo HPC(s)	Spark(s)
300	91	284
600	227	678
1200	478	1330
2000	923	2641

6. CONCLUSIONES: Se logro hacer una implementación efectiva de k-means con la ayuda de sparks y su librería MLib.

La solución hecha en el modulo anterior dio un mejor rendimiento pero hay que tener en cuenta que la solución de big data requirió mucho menos tiempo en implementarla. Esto nos quiere decir 2 cosas.

Primero que una solución manual puede dar un mejor rendimiento. Pero sparks tiene la ventaja de que entrega soluciones mucho más rápido. Si la situación requiere de varias soluciones de big data en tiempo corto Spark es la mejor opción.

Segundo que sparks brindad una cantidad grande de herramientas que agilizan mucho el proceso de desarrollo de producción. El proceso se facilita mucho más con las librerías que brinda sparks.

7.REFERENCIAS:

1. Anna Huang *Similarity measures for text document cluster- ing.* Proceedings of the Sixth New Zealand, April):49–56, 2008.