**TF-IDF Application: Document Search**

**Contents:**

1. **Introduction**
2. **Background**
3. **Technical details**
4. **Performance evaluation**
5. **Conclusion/lessons learnt**
6. **References**

1. **Introduction**

This project is all about document search in large corpus, the application takes query from the user and output will be the relevant document from corpus, related to that query, further more queries can have executed. The query can have single and multiple parameters.

TF-IDF based approached is used which has been usuallypragmatic in information retrieval classifications. It allows us to retrieve documents that are strictly related to a query expression but not aprecise string-match.

1. **Background**

In today’s era data is generating with a high speed in varieties. Storing and managing that data, doing analytics on that and performing any operation on data is a challenging task.

Example, huge number of documents like mails, news, blogs, and articles are on internet and what if somebody wants to search some document from that corpus? Obviously! It is not possible if we don’t have any searching algorithm or any program that execute on corpus and retrieve the relevant document according to search by user.

In a huge corpus it is not possible to get relevant document according to the query until unless any search operation is not performed.

Hence, some algorithm or application is required to find out relevant document like ***blogs, news, articles, stories etc. according to user query***. That filtered out required document from the corpus.

This type of stories is implemented in Google and can be implementing on projects in which huge number of documents are present. This helps in finding or filtering the corpus to get relevant document according to user query.

Use of these types of stories is in blogs, news, articles etc. Hence, websites like Quora, medium, stack overflow, or any news article websites can use this technique to filter out documents.

1. **Technical details**

**Language used:**

* Java (jdk\_1.8).

**Frameworks/technologies used:**

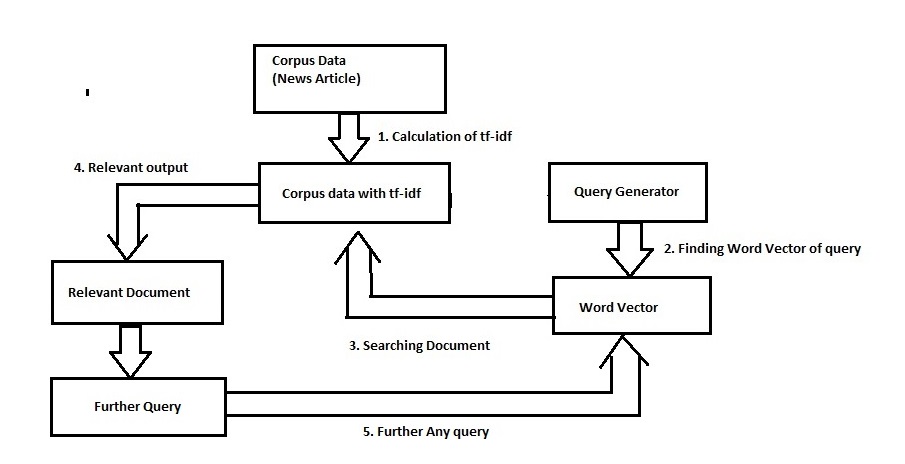
* Hadoop (hdfs): It used to store large corpus in distributed manner:
* Yarn: Used for managing resources and application in cluster.
* Spark: Framework for running parallel processing in cluster.

**Implementation:**

Implementation has been done on Hadoop environment in which Hadoop hdfs is used to store large size of corpus in distributed manner and yarn is used to manage computing or processing units. For figuring out relevant document apache spark framework is used calculating tfidf and then in the corpus according to user query application following process has been used.

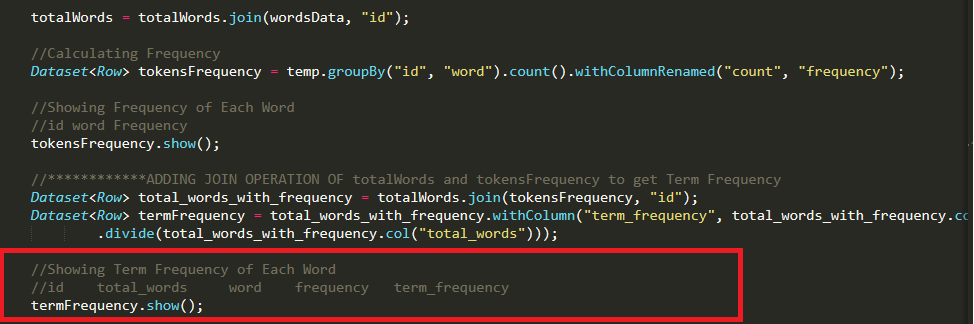
* 1. Calculation of TF-IDF is done using SparkSQL.
  2. Query is taken as input to find relevant corpus and then query running on SparkSQL to search relevant documents for words given in the user query.
  3. Output of relevant document sorted according to tfidf value.
  4. Further query can be done to search further more documents.

**Implementation architecture:** The block diagram of implementation is as follows:



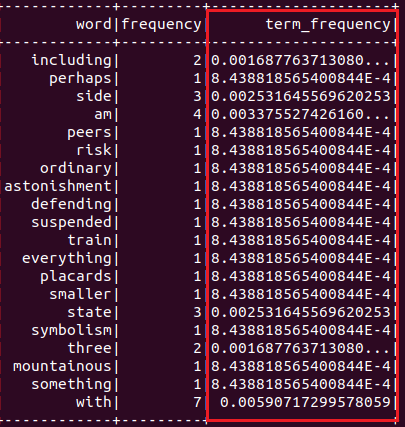
**Implementation Steps are as follows:**

1. Calculations of TF-IDF
2. Query for documents
3. Searching for documents
4. Relevant document from corpus
5. Furthermore, query
   1. **Calculations of TF-IDF:** TF-IDF is being calculated while calculating TF and IDF the process/code and run time consoles are as follows respectively:
      * **Term Frequency calculation code fragment is as:**

****

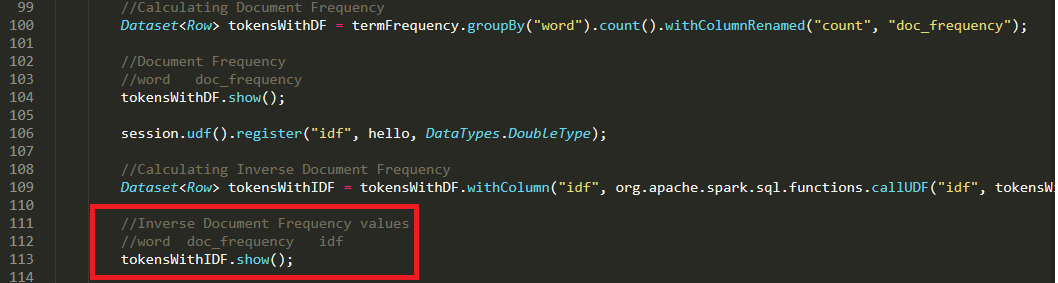
In above code snippet Term Frequency of every single word is calculated by dividing the total number of words in document.

* **And runtime console output of term frequency is as:**

****

Output of Term Frequency is shown in above snap shot where word and its frequency and term frequency is shown.

* **Inverse Document Frequency calculation code fragment**

****

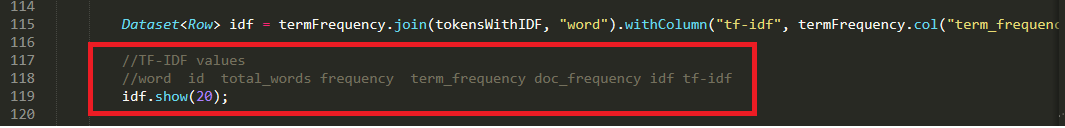
In above code snippet IDF is calculated using the UDF function and tokensWithIDF.show will print below output.

* **And runtime console output of Inverse Document Frequency is as:**



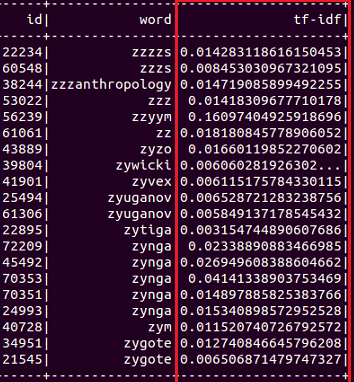
Above code snippet shows the doc\_frequency and inverse Document Frequency (idf) for every single word.

* **TF-IDF calculation is as:**



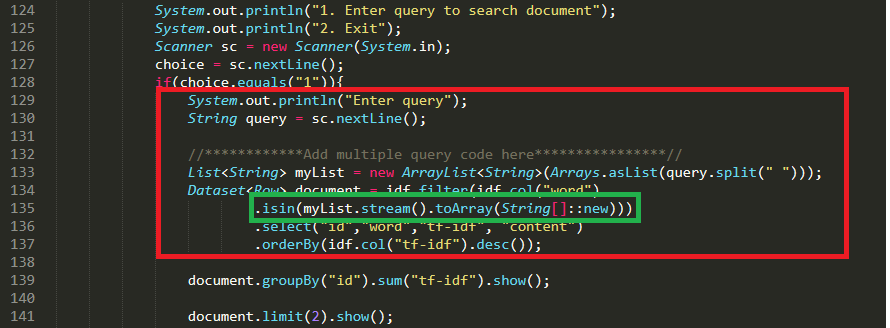
In above code snippet TF-IDF is calculated by multiplying TF and IDF calculated in the previous steps and TF-IDF is showing using the idf. show() statement.

* **And runtime console output of TF-IDF is as:**

****

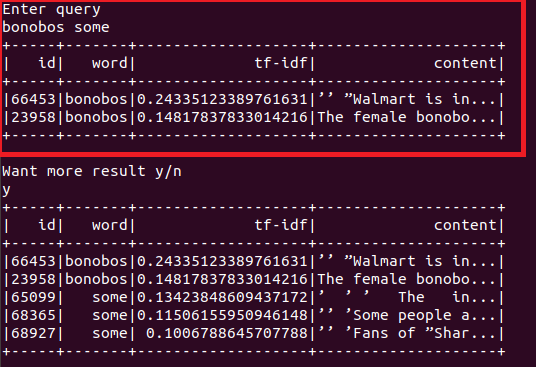
In above output snippet TF-IDF value of every single word with its id is shown.

1. **Query for documents:** Document search code is as:

****

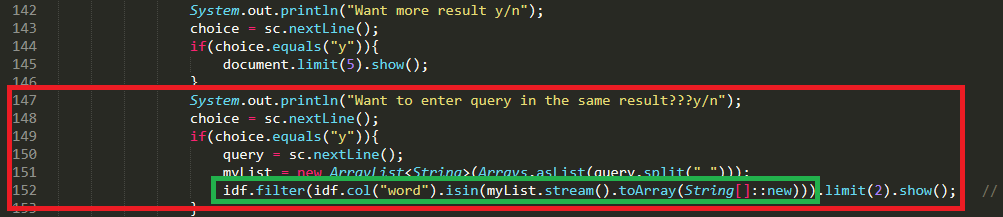
Above code snippet is used to take query of multiple words as shown in green lined rectangle and search the relevant document in which query words occur ***isin()*** is used to search multiple words.

* **Query search result runtime output is as:**



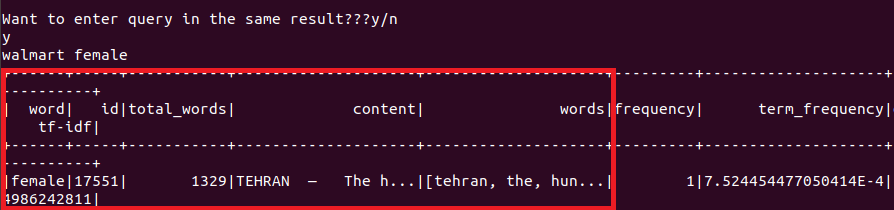
Above output snippet shows the result of searched query, on the basis of TF-IDF value relevant documents are extracted e.g. document id 66453 and 65099 are relevant documents according to query ‘bonobos some’ while showing maximum TF-IDF value which is its parameter of measure with other attributes.

1. **Searching for documents:** Documents are being searched based upon maximum TF-IDF as shown in above screen shot sorted in decreasing (TF-IDF column) order e.g. 66453, and 23958 documents have been returned in above screen shot.
2. **Relevant document from corpus:** Based upon searching criteria in above clause 3 the most relevant document will be returned based highest TF-IDF.
3. **Furthermore, query:** In this step we have provided option to search more document from the same query i.e. from the same spark SQL dataset as shown below:

****

In above code snippet it’s asking for further more query on the searched result and return the result from previous searched document.

* And the further more query option (above code fragment) is as:



Result is shown on the basis of query ‘walmart female’ which shows the result on the basis of previous query search results.

1. **Performance evaluation:** Following experimental setup is created for executing this project:

**Experimental setup:** We have created a cluster of 3 nodes (given virtual machines) i.e. namenode, datanode1, and datanode2, as a virtual machines depicted as:

namenode: Main Server

datanode2

datanode1

**Namenode:**

Namenode is main node in our application cluster, where we have installed apache Hadoop and spark, wherein Hadoop is being used for storage of corpus data and spark is used for data processing and analysis.

**Datanode1 & Datanode2:**

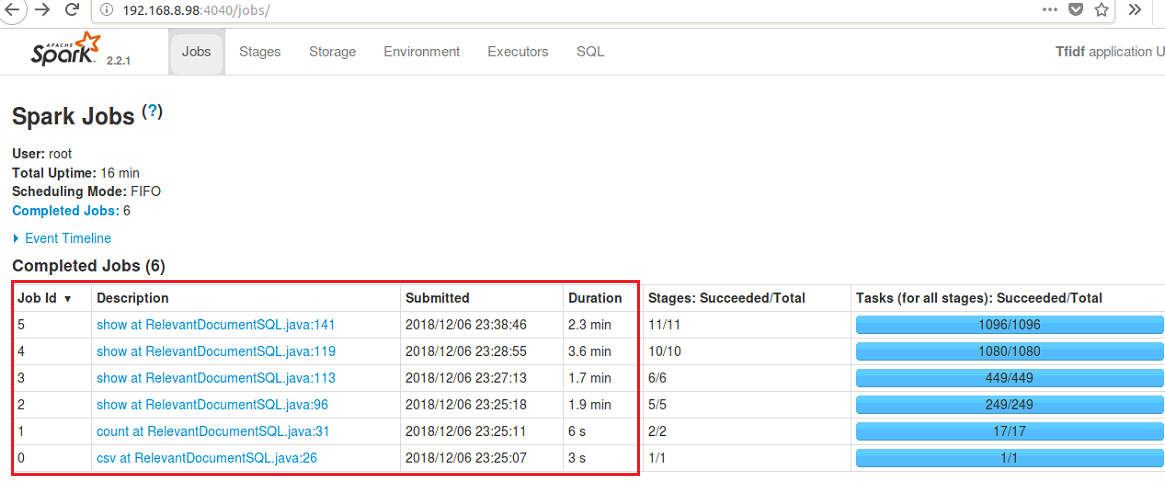
These are data nodes which will be used by **namenode** to do Hadoop’s/spark’s operations i.e. for data storage, processing, and analytics.

**Dataset:** Following data set has been used**:**

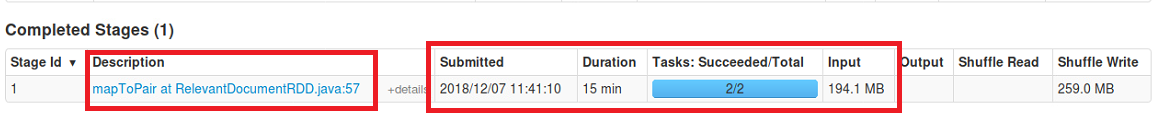
* A csv corpus of all the news 1,43,000 articles from 15 American publications.
* Link to the dataset is: <https://www.kaggle.com/snapcrack/all-the-news>

**Results:** Performance is evaluated by taking a reference of **spark’s Web UI** where statistics of every process is shown.

* **Performance statistics with Spark SQL:** Statistics of Spark SQL process on Spark Web UI.



* **Performance statistics with Spark RDD:** Statistics of Spark RDD process on Spark Web UI.

**Result:**

From above 2 images it is seen that time taken by spark RDD is more compare to Spark SQL.

The line RelevantDocumentRDD.java:57 is­­­­ for word count which is taking 15 minutes to complete and similar line in Spark SQL is Relevant Document SQL.java:96 which is taking only 1.9 Minutes to complete the word count.

Hence, for word count task it seems that Spark SQL is faster than Spark RDD in this use case. Similarly, Term Frequency and Inverse Document frequency will also be faster in Spark SQL which makes total execution faster in Spark SQL, hence we have used spark SQL APIs for this project.

**Reason:**

It depends on dataset and user story, that which spark platform should be used to make process faster, as per TF-IDF use case according to above results it seems Spark SQL is much faster than Spark RDD.

1. **Conclusion and Lessons Learnt**

**Conclusion:**

The TF-IDF is weight of every single word in a corpus which is used to extract relevant document from the corpus. To calculate TF-IDF of large corpus we need fast processing that can be achieved by parallel computing using yarn or spark which are frameworks that provide parallel computing on a cluster. Hence, news article is stored in cluster (HDFS) and analyzed using parallel processing.

**Lessons Learnt:**

* 1. Big Data tools that are capable of handling and processing structured and unstructured data on parallel computing (cluster).
* **Hadoop:** Provides Distributed storage by forming cluster.
* **Yarn and MapReduce:** Provides parallel processing on large amount of data stored on hadoop cluster.
* **Spark:** Spark is an on memory processing engine which process data in memory that makes it 100 times faster than normal program.
* **Spark (SQL):** Text processing is done in spark SQL.
  1. **Text Processing:** Before analyzing data it is necessary to process the data sets i.e.
  + **Cleaning and Munging** the datasets to remove the null values or to add some arbitrary values to make dataset analyzable.
  + **Removing stop words** that is not useful in analyzing data.

1. **References:**
   1. <https://en.wikipedia.org/wiki/Tf-idf>
   2. <http://www.tfidf.com/>
   3. <https://lizrush.gitbooks.io/algorithms-for-webdevs-ebook/content/chapters/tf-idf.html>
   4. <https://spark.apache.org/>