**TF-IDF**

**Abstract:**

Calculating TF-IDF from a corpus using Hadoop for distributed storage and Yarn for parallel Processing.

**Introduction:**

Tf-idf stands for *term frequency-inverse document frequency*, and the tf-idf weight is a weight often used in information retrieval and text mining. This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus. Variations of the tf-idf weighting scheme are often used by search engines as a central tool in scoring and ranking a document's relevance given a user query.

TF-IDF can be successfully used for stop-words filtering in various subject fields including text summarization and classification.

**Application of TF-IDF:**

This algorithm is useful when you have a document set, particularly a large one, which needs to be categorized. It is especially nifty because you don't need to train a model ahead of time and it will automatically account for differences in lengths of documents.

TF-IDF is use to extract the relevant document from corpus where the document specification OR categorization of document is not defined in the corpus.

This is helpful if we have lots of blogs in our website or documents but tag of that blog/doc or type of that blog/doc is not defined than to categorized all the blog/doc and to make efficient searching of relevant document we calculate TF-IDF of each word of that document. Which helps to retrieve the relevant document/blogs.

**The Math:**

Typically, the TF-IDF weight is composed by two terms: the first computes the normalized Term Frequency (TF), aka. the number of times a word appears in a document, divided by the total number of words in that document; the second term is the Inverse Document Frequency (IDF), computed as the logarithm of the number of the documents in the corpus divided by the number of documents where the specific term appears.

* **TF: Term Frequency**, which measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. Thus, the term frequency is often divided by the document length (aka. the total number of terms in the document) as a way of normalization:

TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document).

* **IDF: Inverse Document Frequency**, which measures how important a term is. While computing TF, all terms are considered equally important. However, it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance. Thus, we need to weigh down the frequent terms while scale up the rare ones, by computing the following:   
    
  IDF(t) = loge (Total number of documents / Number of documents with term t in it).

**Motivational Example:**

### **Generate Scores for Each Document:**

Let's say you have a 100-word blog post with the word "JavaScript" in it 5 times. The calculation for the Term Frequency would be:

TF = 5/100 = 0.05

Next, assume your entire collection of blog posts has 10,000 documents and the word "JavaScript" appears at least once in 100 of these. The Inverse Document Frequency calculation would look like this:

IDF = loge (10,000/100) = 2

To calculate the TF-IDF, we multiply the previous two values. This gives us the final score:

TF-IDF = 0.05 \* 2 = 0.1

**Implementation:**

As discussed above TF-IDF is used for retrieving relevant document from a corpus or use for text processing.

As we have to process TF-IDF in corpus hence Hadoop cluster is needed for storing that huge number of documents and process it using MapReduce which perform distributing computing.

**Hadoop (Introduction):**

## **Problems with Traditional Approach:**

In traditional approach, the main issue was handling the heterogeneity of data i.e. structured, semi-structured and unstructured. The RDBMS focuses mostly on structured data. And Hadoop specializes in semi-structured, unstructured data like text, videos, audios, Facebook posts, logs, etc.

* ***The first problem is storing the colossal amount of data.***

Storing this huge data in a traditional system is not possible. The reason is obvious, the storage will be limited only to one system and the data is increasing at a tremendous rate.

### ***Second problem is accessing and processing speed***.

The hard disk capacity is increasing but the disk transfer speed or the access speed is not increasing at similar rate. If you have only one 100 Mbps I/O channel and you are processing 1TB of data, it will take around 2.91 hours. Now, if you have four machines with one I/O channel, for the same amount of data it will take 43 minutes approx. Thus, accessing and processing speed is the bigger problem than storing Big Data.

## **What is Hadoop?**

Hadoop is a framework that allows you to first store Big Data in a distributed environment, so that, you can process it parallelly.

### ***The first problem is storing Big data***.

HDFS provides a distributed way to store Big data. Your data is stored in blocks across the Data Nodes and you can specify the size of blocks.

It also solves the scaling problem. It focuses on ***horizontal scaling*** instead of vertical scaling. You can always add some extra data nodes to HDFS cluster as and when required, instead of scaling up the resources of your Data Nodes.

* ***Next problem was storing the variety of data***.

With HDFS we can store all kinds of data whether it is structured, semi-structured or unstructured.

### ***Third challenge was accessing & processing the data faster***.

Processing is done parallelly in distributed computing which increases the accessing and processing power.

**MapReduce (Introduction):**  
Hadoop MapReduce (Hadoop Map/Reduce) is a software framework for distributed processing of large data sets on computing clusters.

The basic unit of information, used in MapReduce is a (Key, value) pair. All types of structured and unstructured data need to be translated to this basic unit, before feeding the data to MapReduce model.

* **Mapper:**

Mapper task is the first phase of processing that processes each input record (from Record Reader) and generates an intermediate key-value pair. Hadoop Mapper store intermediate-output on the local disk. Hence, in this phase, we specify all the complex logic/business rules/costly code.

* **Reducer:**

Reducer takes the output of the [Mapper](http://data-flair.training/blogs/mapper-in-hadoop-mapreduce/) (intermediate key-value pair) process each of them to generate the output. The output of the reducer is the final output, which is stored in HDFS. Usually, in the Hadoop Reducer, we do aggregation or summation sort of computation.

**Hadoop and MapReduce Configuration:**

Hadoop and MapReduce configuration and setup procedure are done in separate file named as Hadoop\_setup in directory.

**Technologies to be used:**

* Hadoop (Distributing Storage)
* MapReduce (Distributed Computing and Parallel Processing)
* Yarn (A Hadoop tool)
* Java

**Workflow of MapReduce Program for calculating TF-IDF:**

\*\*Java files that are mention here are found in Program Files Directories.

* **Step 1:** Word count of individual files:

In WordCount.java file Mapper Reducer program is written that counts the frequency of word of individual file and store the word with its frequency in its corresponding file in wordcount directory(Stored in hdfs cluster) with the same file name as processed by mapper reducer.

Key used in mapper program is “file name” and corresponding value of key used is “word” consisting in file.

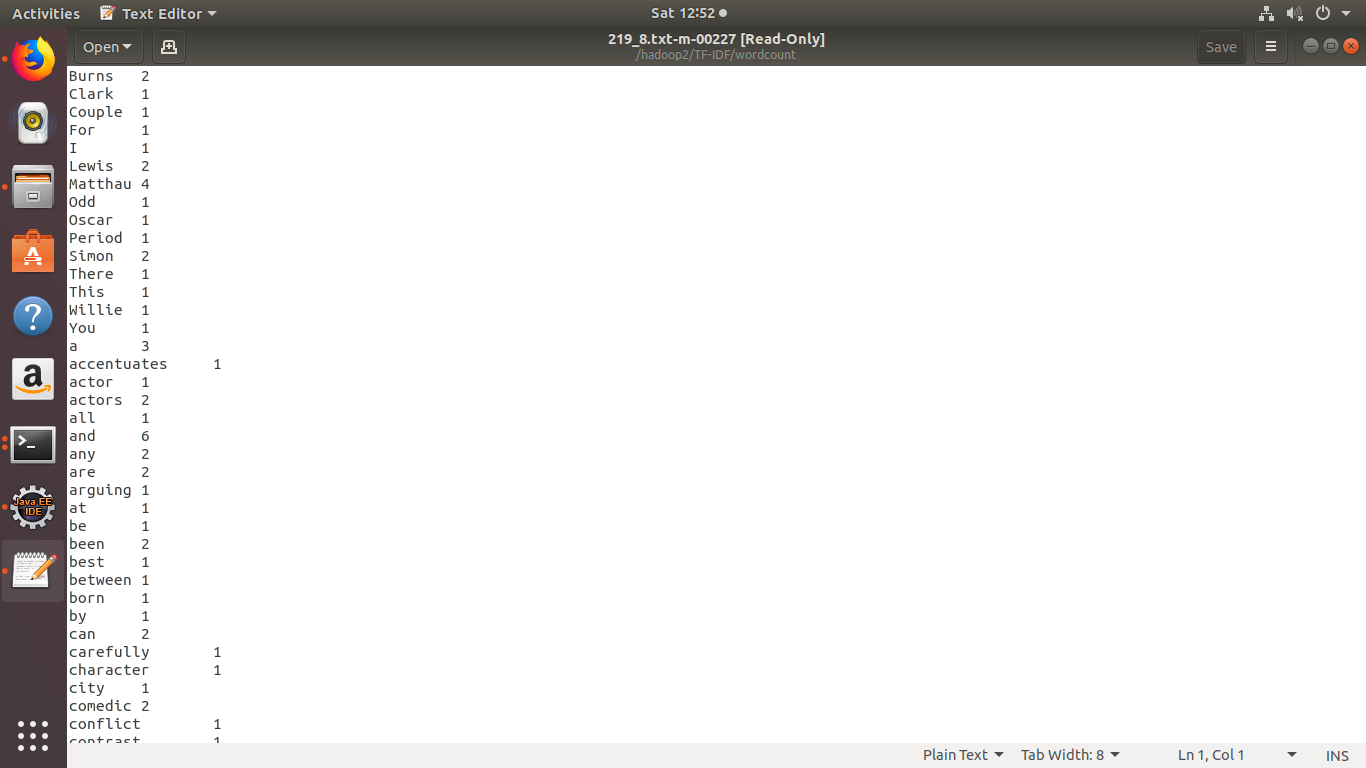
**Output of the wordcount directory files are like**:

221\_8.txt-m-00010

Retain 2

Retailer 4

Where Retain is word of document 221\_8.txt-m-00010 and 2 is its frequency in that document.



* **Step 2:** Calculating Term frequency of documents:

For calculating term frequency, word count directory is used where files have their word with frequency. Using these files (Which consists frequency of each word) term frequency is calculated by giving input wordcount directory to TermFreq.java file and output will be the directory having files with term frequency of every single word consisting in it.

**Key Value pair used:**

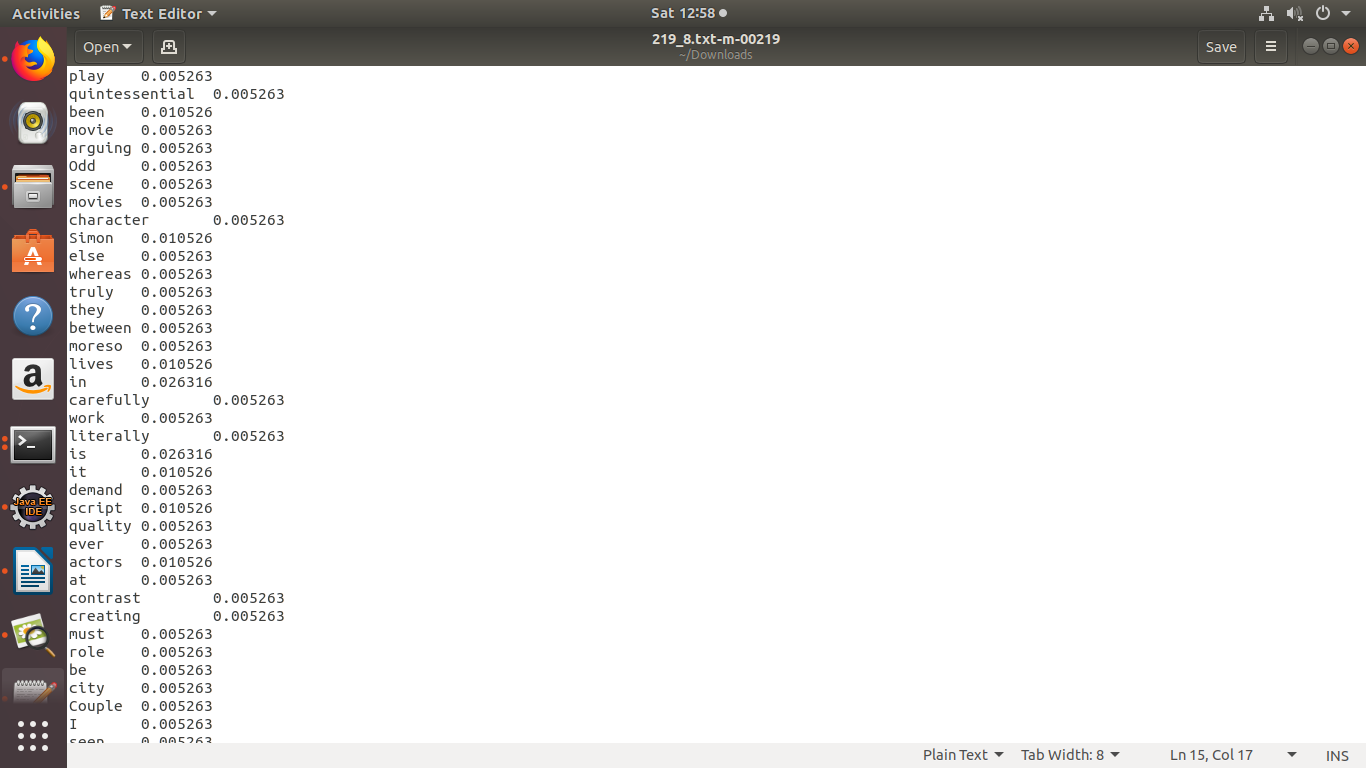
Key used in mapper is “filename” and value used is word concatenation

with its frequency and this key value pair is transferred to reducer. Where Total number of words in particular document is counted and term frequency is calculated for each word.

**Formula used:**

Term Frequency = Occurrence of unique word in document/total number of words in document.

**Output of the Term Frequency File are like:**



* **Step 3:** **Calculating TF-IDF of document:**

For calculating TF-IDF, Term Frequency file is used as an input of InvDocFreq.java and output directory is created named as tf\_idf where all the files are consists having their tf-idf value in it, corresponding to every word in each file.

**Key Value pair used:**

In mapper program, key is filename concatenation with term frequency of word that mapper takes as an input from record reader and value is the word of that corresponding file that mapper takes as an input. And this key value pair is transferred to reducer.

Example: of key value pair that mapper generate

221\_8.txt-m-00010

Retain 0.0021254

Retailer 0.0523644

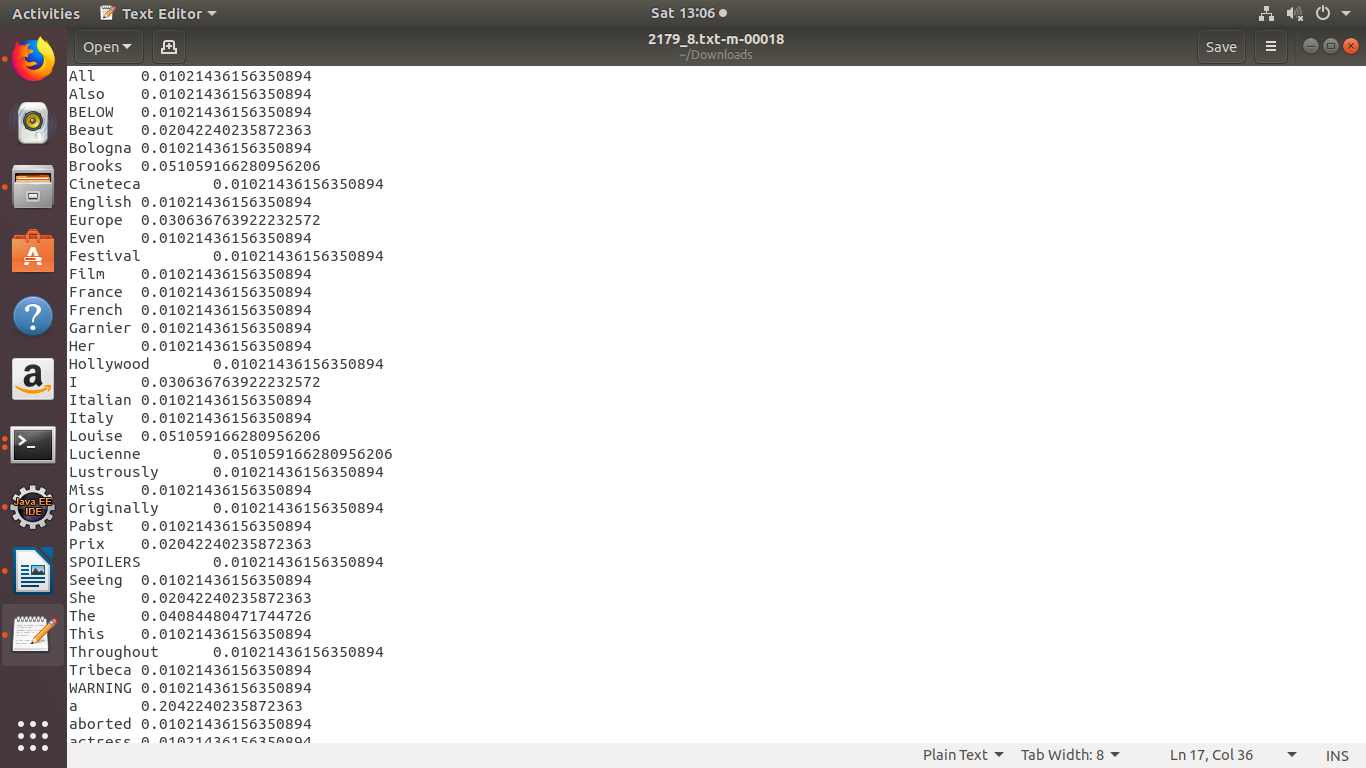
Let’s suppose above is a data of term frequency which is given to mapper as an input than mapper will form key:” 221\_8.txt-0.0021254” with corresponding value “Retain” and “221\_8.txt-0.0523644” as second key with corresponding value “Retailer” and so on for evey file.

**Output of the Term Frequency Inverse Document Frequency File are like:**

221\_8.txt-m-00010

Retain 0.0002489

Retailer 0.0001542



**Results:**

**Results of every file are put in results directory where the name of file defined the respective results.**

**Assumption:**

We have preprocessed the corpus text file to calculate TF-IDF value so that we don’t need to process the whole corpus again and again.

**References:**

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