Experiment 10 - Batch Analysis using Spark

| Roll No. |  |
| --- | --- |
| Name |  |
| Class | D15A |
| Subject | DS using Python Lab |
| LO Mapped | LO5: Design and Build an application that performs exploratory data analysis using Apache Spark |
|  |  |

**Aim**:

To perform Batch Data Analysis using Spark

**Introduction**:

MapReduce is a programming paradigm that enables massive scalability across hundreds or thousands of servers in a Hadoop cluster. As the processing component, MapReduce is the heart of Apache Hadoop. The term "MapReduce" refers to two separate and distinct tasks that Hadoop programs perform. The first is the map job, which takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs).

The reduce job takes the output from a map as input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce job is always performed after the map job.

MapReduce programming offers several benefits to help you gain valuable insights from your big data:

1. Scalability - Businesses can process petabytes of data stored in HDFS
2. Flexibility - Hadoop Enables easier access to multiple data sources and types of data.
3. Speed - With parallel processing and minimal data movement, large amounts of data can be processed quickly.

The major advantage of MapReduce is that it is easy to scale data processing over multiple computing nodes. Under the MapReduce model, the data processing primitives are called mappers and reducers. Decomposing a data processing application into mappers and reducers is sometimes nontrivial. But, once we write an application in the MapReduce form, scaling the application to run over hundreds, thousands, or even tens of thousands of machines in a cluster is merely a configuration change. This simple scalability is what has attracted many programmers to use the MapReduce model.

**Dataset Collection**

The datasets used here are

1. Wikipedia article

Format: Text file  
Preprocessing:  
A. The text file is loaded using SparkContext textFile method

B. Remove Punctuation and Transform All Words to Lowercase.

C. We use split function to separate the words in all lines .

D. We do a filtering below to exclude whitespaces.

1. Song Lyrics Dataset

Billboard has published a Year-End Hot 100 every December since 1958. The chart measures the performance of singles in the U.S. throughout the year. Using R, I’ve combined the lyrics from 50 years of Billboard Year-End Hot 100 (1965-2015) into one dataset for analysis.

**Approach**:

Approach to count the words using Spark:

1. Let's create an RDD by using the following command

***data = sc.textFile("file\_name.txt")***

1. Here, pass any filename that contains the data. Now, we can read the generated result by using the following command.

***data.collect***

1. Here, we split the existing data in the form of individual words by using the following command.

***splitdata= book.flatMap(lambda x: x.split()).countByValue()***

1. Now, we can read the generated result by using the following command.

***splitdata.collect***

1. Now, perform the map operation.

***for i, (word, count) in enumerate(word\_counts.items()):***

***if i == 100: break***

***print(word, count)***

Here, we are assigning a value 1 to each word. Now, we can read the generated result by running the for loop.

1. Now, perform the reduce operation if needed.

***reducedata = mapdata.reduceByKey(lambda a,b : a+b)***

Here, we are summarizing the generated data.

**Implementation**:

**Setup**

!pip install pyspark

!pip install -U -q PyDrive

!apt install openjdk-8-jdk-headless -qq

!wget -q https://dlcdn.apache.org/spark/spark-3.2.1/spark-3.2.1-bin-hadoop3.2.tgz

!tar xf spark-3.2.1-bin-hadoop3.2.tgz

**Setting Environment Variables**

import os

os.environ["JAVA\_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"

os.environ["SPARK\_HOME"] = "/content/spark-3.2.1-bin-hadoop3.2"

os.environ["PYTHONPATH"] = "%SPARK\_HOME%\python;%SPARK\_HOME%\python\lib\py4j-0.10.9.3-src.zip:%PYTHONPATH%"

**Setting the SparkContext**

from pyspark import SparkConf, SparkContext

conf = SparkConf().setMaster("local").setAppName("word-counts")

sc = SparkContext(conf=conf)

**Setting up the data**

article = sc.textFile("Machine\_Learning\_Wikipedia.txt")

**Preprocessing**

def lower\_clean\_str(x):

punc='!"#$%&\'()\*+,./:;<=>?@[\\]^\_`{|}~-'

lowercased\_str = x.lower()

for ch in punc:

lowercased\_str = lowercased\_str.replace(ch, '')

return lowercased\_str

article = article.map(lower\_clean\_str)

article=article.flatMap(lambda satir: satir.split(" "))

article = article.filter(lambda x:x!='')

**Getting Word Count**

article\_count=article.map(lambda word:(word,1))

article\_count\_RBK=article\_count.reduceByKey(lambda x,y:(x+y)).sortByKey()

article\_count\_RBK=article\_count\_RBK.map(lambda x:(x[1],x[0]))

article\_count\_RBK.sortByKey(False).take(10)

**Loading song lyrics dataset**

import sys

from operator import add

from pyspark.sql import SparkSession

from pyspark.ml.feature import Tokenizer

from pyspark.ml.feature import StopWordsRemover

import pyspark.sql.functions as f

spark = SparkSession\

.builder \

.appName("PythonWordCount") \

.getOrCreate()

data = spark.read.format('csv').options(header='true', inferSchema='true') \

.load('billboard\_lyrics\_1964-2015.csv') \

print('############ CSV extract:')

data.show()

# Count and group word frequencies on the column Lyrics, when splitted by space comma

data.withColumn('word', f.explode(f.split(f.col('Lyrics'), ' '))) \

.groupBy('word') \

.count() \

.sort('count', ascending=False) \

.show()

# To remove stop words (like "I", "The", ...), we need to provide arrays of words, not strings. Here we use APache Spark Tokenizer to do so.

# We create a new column to push our arrays of words

tokenizer = Tokenizer(inputCol="Lyrics", outputCol="words\_token")

tokenized = tokenizer.transform(data).select('Rank','words\_token')

print('############ Tokenized data extract:')

tokenized.show()

# Once in arrays, we can use the Apache Spark function StopWordsRemover

# A new column "words\_clean" is here as an output

remover = StopWordsRemover(inputCol='words\_token', outputCol='words\_clean')

data\_clean = remover.transform(tokenized).select('Rank', 'words\_clean')

print('############ Data Cleaning extract:')

data\_clean.show()

# Final step : like in the beginning, we can group again words and sort them by the most used

result = data\_clean.withColumn('word', f.explode(f.col('words\_clean'))) \

.groupBy('word') \

.count().sort('count', ascending=False) \

print('############ TOP20 Most used words in Billboard songs are:')

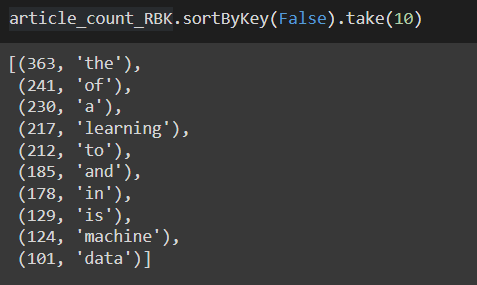
result.show()

# Stop Spark Process

spark.stop()

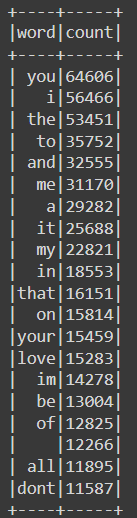
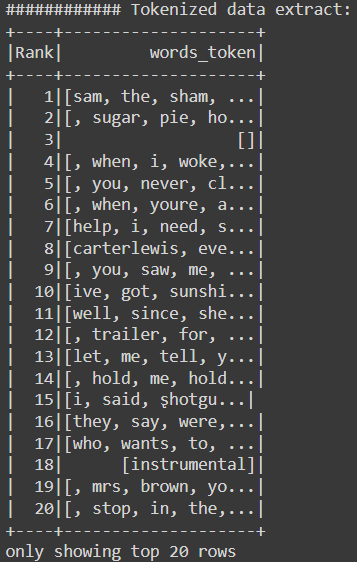
**Results**:

Article:



Song lyrics:



**Conclusion**:

Thus, we have learnt what batch processing is and also learnt how to implement it using Spark.