Big data & analytics Lab

Department of Computer Engineering
National Institute of Technology, Kurukshetra
Haryana, 136119

Lab Record

Rakesh Ranjan
11912050
CS-3(A)

Table of Content

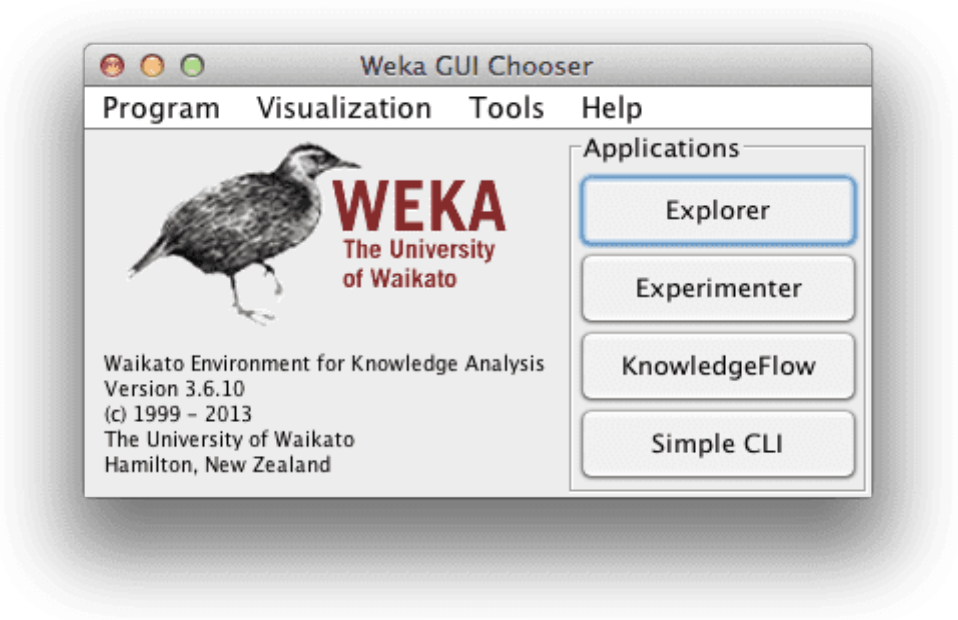
Weka

Introduction:

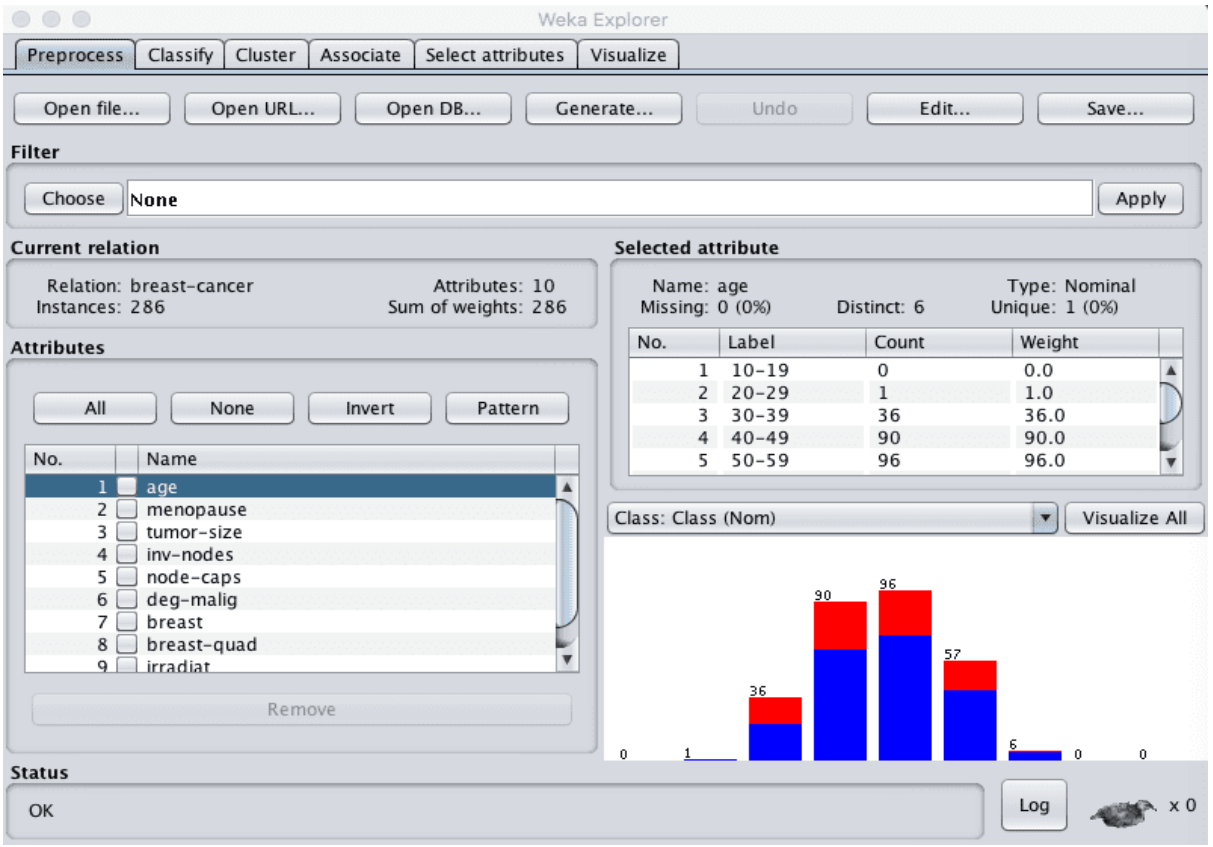
Weka contains a collection of visualisation tools and algorithms for data analysis and predictive modelling, together with graphical user interfaces for easy access to these functions.

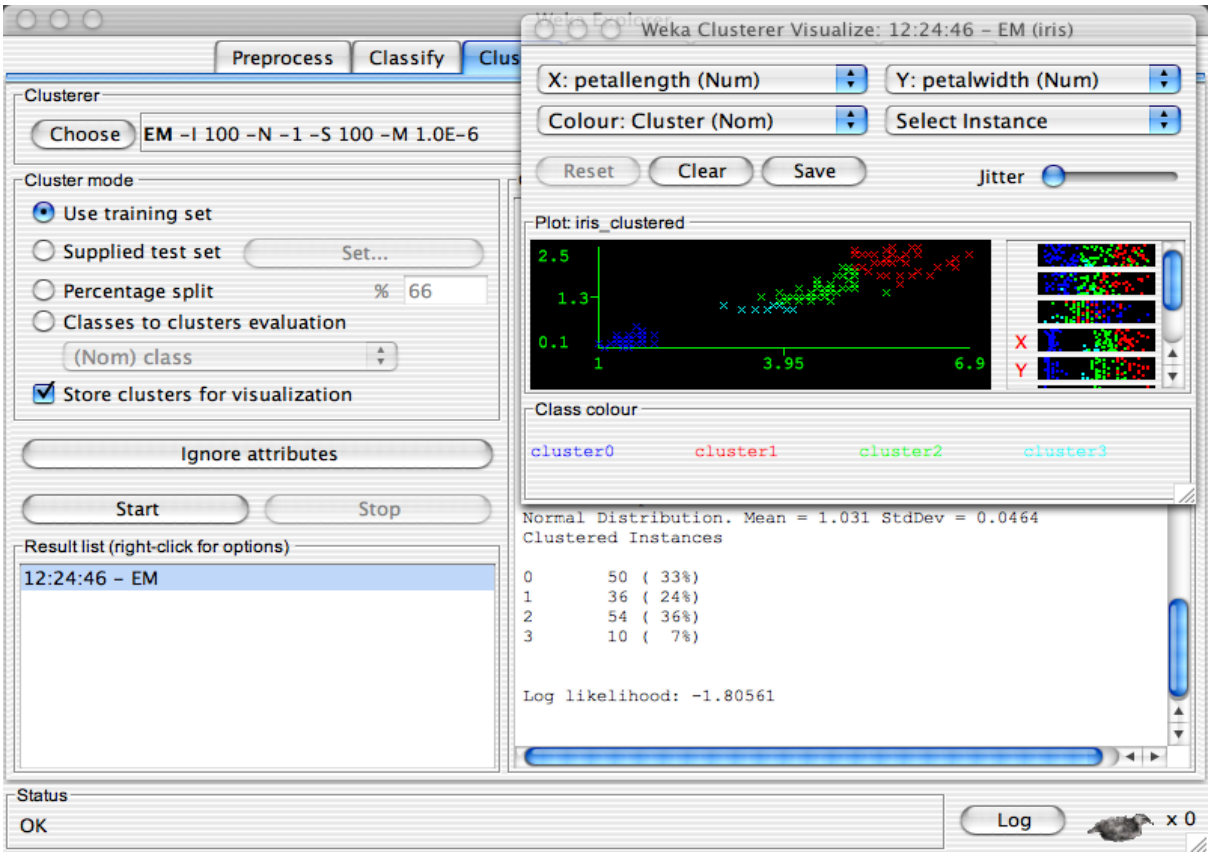
Setup & Installation :

1. Download from the link : https://sourceforge.net/projects/weka/files/latest/
2. Then click on next and next and finally it will get installed.



Experiments:





Hadoop

Introduction: Hadoop is an Apache open source framework written in java that allows distributed processing of large datasets across clusters of computers using simple programming models. The Hadoop framework application works in an environment that provides distributed storage and computation across clusters of computers. Hadoop is designed to scale up from a single server to thousands of machines, each offering local computation and storage.

Installation and Setup :

1. Download hadoop : https://hadoop.apache.org/releases.html
2. Download Jre & Jdk
3. Set the path of hadoop home directory to the system environmental variable, similarly set the path of jdk
4. Now start your hadoop by the command : start-all.cmd & for closing hadoop use : stop-all.cmd

Experiments :

1. Word Count

Command :
hadoop jar /path/to/wordcount-1.0-SNAPSHOT.jar /path/to/inputDir /pathToOutputDir

Input Console:

```
[ec2-user@ip-172-31-32-205 bin]$ cd /home/ec2-user/data/input/
[ec2-user@ip-172-31-32-205 input]$ ls
input1.txt  input2.txt  input3.txt
[ec2-user@ip-172-31-32-205 input]$ more input1.txt
Hadoop is the Elephant King!
A yellow and elegant thing.
He never forgets
Useful data, or lets
An extraneous element cling!
[ec2-user@ip-172-31-32-205 input]$ more input2.txt
A wonderful king is Hadoop.
The elephant plays well with Sqoop.
But what helps him to thrive
Are Impala, and Hive,
And HDFS in the group.
[ec2-user@ip-172-31-32-205 input]$ more input3.txt
Hadoop is an elegant fellow.
An elephant gentle and mellow.
He never gets mad,
Or does anything bad,
Because, at his core, he is yellow.
[ec2-user@ip-172-31-32-205 input]$
```

Output Console:

```
[ec2-user@ip-172-31-32-205 output]$ pwd
/home/ec2-user/data/output
[ec2-user@ip-172-31-32-205 output]$ ls -lah
total 12K
drwxr-xr-x. 2 ec2-user ec2-user  84 Feb 15 03:11 .
drwxrwxr-x. 4 ec2-user ec2-user  33 Feb 15 03:11 ..
-rw-r--r--. 1 ec2-user ec2-user 458 Feb 15 03:11 part-00000
-rw-r--r--. 1 ec2-user ec2-user  12 Feb 15 03:11 .part-00000.crc
-rw-r--r--. 1 ec2-user ec2-user   0 Feb 15 03:11 _SUCCESS
-rw-r--r--. 1 ec2-user ec2-user   8 Feb 15 03:11 ._SUCCESS.crc
[ec2-user@ip-172-31-32-205 output]$
```

```
[ec2-user@ip-172-31-32-205 output]$ less part-00000
A       2
An      2
And     1
Are     1
Because,        1
But     1
Elephant        1
HDFS    1
Hadoop  2
Hadoop. 1
He      2
Hive,   1
Impala, 1
King!   1
Or      1
Sqoop.  1
The     1
Useful  1
an      1
and     3
anything        1
at      1
bad,    1
cling!  1
core,   1
data,   1
does    1
elegant 2
element 1
elephant        2
extraneous      1
fellow. 1
forgets 1
gentle  1
gets    1
group.  1
he      1
helps   1
him     1
his     1
in      1
```

2. Avg student marks:

Student mark dataset:

```
Administrator: Command Prompt
G:\hadoop\sbin>hadoop fs -put G:\Hadoop_Experiments\NameList.csv /input_average

G:\hadoop\sbin>hadoop fs -ls /input_average
Found 1 items
-rw-r--r--   1 hp supergroup        202 2021-04-06 00:32 /input_average/NameList.csv

G:\hadoop\sbin>hadoop dfs -cat /input_average/NameList.csv
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.
130581,A,9.2
130583,B,8.4
130582,C,8.2
130585,D,9
130584,E,7.8
130585,D,8.8
130584,E,8
130583,B,8.6
130582,C,8.2
130581,A,9
130581,A,8.6
130583,B,8.2
130585,D,8.4
130582,C,8
130584,E,8.8

G:\hadoop\sbin>
```

Jar file:

```
eclipse-workspace - AverageMapReduce/src/com/mapreduce/average/AverageCalculator.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer
AverageMapReduce
  JRE System Library [JavaSE-1.8]
  src
    com.mapreduce.average
      AverageCalculator.java
  Referenced Libraries
KmeansMapReduce
MapReduceKmeansProject
MapReduceProject
MapReduceWordCount

AverageCalculator.java
1  package com.mapreduce.average;
2
3  import org.apache.hadoop.conf.Configuration;
16
17 /**
18  * @author Keet Malin Sugathadasa
19  * Distributed Systems Homework 2
20  * MapReduce Assignment
21  *
22  */
23 public class AverageCalculator {
24
25     //Driver Class
26     public static void main(String[] args) throws Exception {
27         //set up configurations
28         Configuration c = new Configuration();
29         String[] files = new GenericOptionsParser(c, args).getRemainingArgs();
30         Path input = new Path(files[0]);
31         Path output = new Path(files[1]);
32         Job j = new Job(c, "averageCalculator");
33         j.setJarByClass(AverageCalculator.class);
34         j.setMapperClass(MapForAverage.class);
35         j.setReducerClass(ReduceForAverage.class);
36         j.setOutputKeyClass(Text.class);
37         j.setOutputValueClass(FloatWritable.class);
38
39         //get input paths from arguments
40         FileInputFormat.addInputPath(j, input);

Outline
com.mapreduce.average
  AverageCalculator
    main(String[]) : void
    MapForAverage
    ReduceForAverage

Problems  Javadoc  Declaration
0 errors, 25 warnings, 0 others
Description | Resource | Path | Location | Type
Warnings (25 items)
```

Output Console:

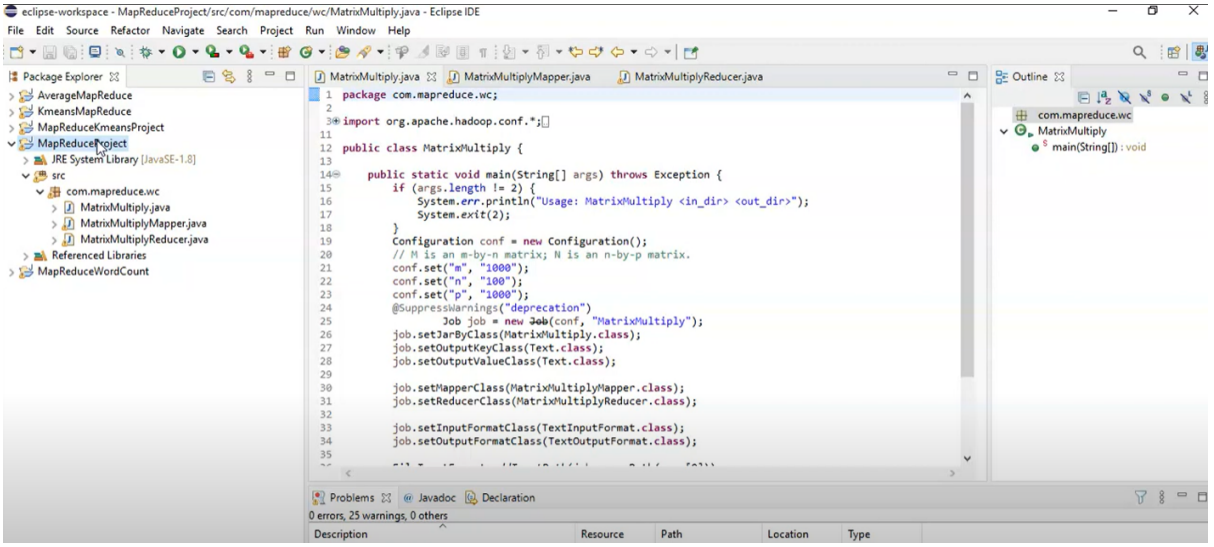
```
Administrator: Command Prompt
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=202
        File Output Format Counters
                Bytes Written=70
Time Elapsed : 46098

G:\hadoop\sbin>hadoop dfs -cat /output_dir_average/*
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.
130581  8.933333
130582  8.133333
130583  8.4
130584  8.2
130585  8.733333

G:\hadoop\sbin>
```

3. Matrix Multiplication:

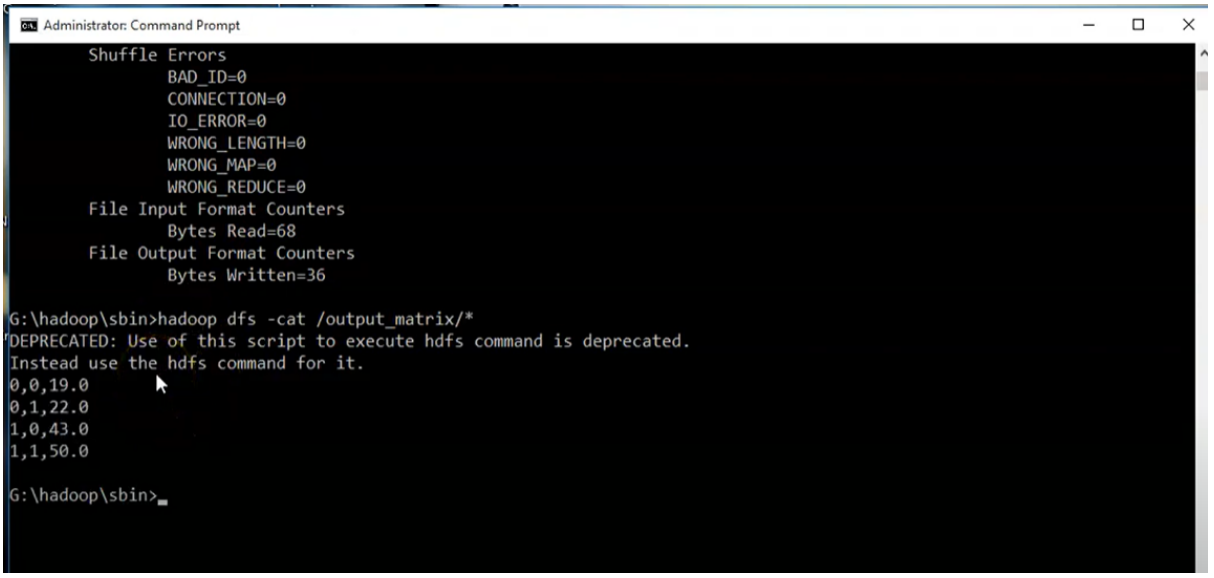
Inputting data of matrix:

```
Administrator: Command Prompt
INFO: No tasks running with the specified criteria.

G:\hadoop\sbin>start-all.cmd
This script is Deprecated. Instead use start-dfs.cmd and start-yarn.cmd
starting yarn daemons

G:\hadoop\sbin>hadoop fs -mkdir /input_matrix

G:\hadoop\sbin>hadoop fs -put G:\Hadoop_Experiments\MatrixMultiply_M.txt /input_matrix

G:\hadoop\sbin>hadoop fs -ls /input_matrix
Found 1 items
-rw-r--r--   1 hp supergroup         34 2021-04-06 00:07 /input_matrix/MatrixMultiply_M.txt

G:\hadoop\sbin>hadoop dfs -cat /input_matrix/MatrixMultiply_M.txt
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.
M,0,0,1
M,0,1,2
M,1,0,3
M,1,1,4
G:\hadoop\sbin>
```

Jar File:



Output Console:



Spark

Introduction: Spark is a lightning-fast cluster computing technology, designed for fast computation. It is based on Hadoop MapReduce and it extends the MapReduce model to efficiently use it for more types of computations, which includes interactive queries and stream processing.

Setup & Installation :

1. Follow this tutorial for installation
   https://sparkbyexamples.com/spark/apache-spark-installation-on-windows/

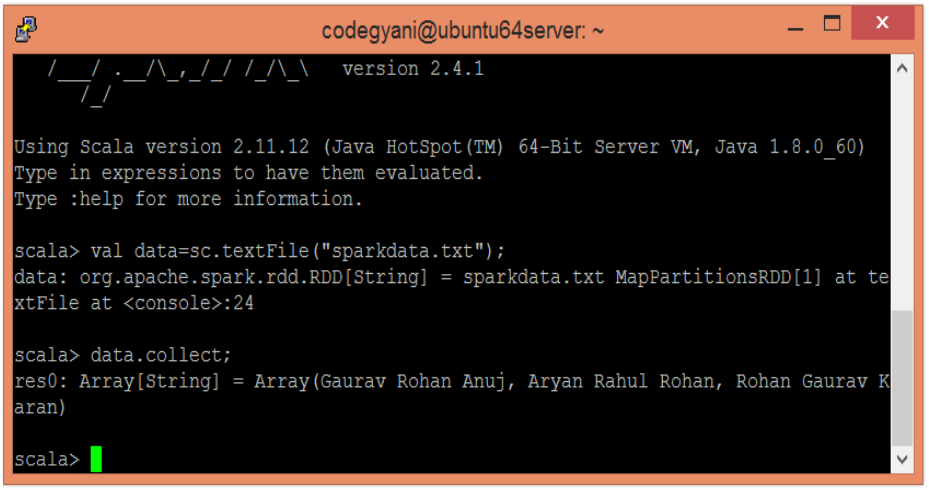
Experiments :

1. Word Count:

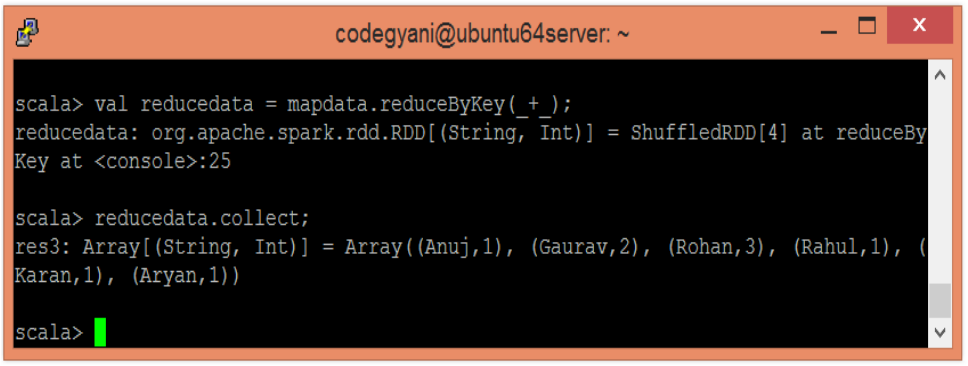
Code:

```
val baseRDD = sc.textFile("word_count_data.txt")
val splitData = baseRDD.flatMap(line=>line.split(" "));
splitData.collect;

val mapData= splitData.map(value=>(value,1))
val reduceData = mapData.reduceByKey(_+_);
```

Input :



Output:



2. Average Marks:

Code:

```
val baseRDD = sc.textFile("avg_marks_dataSet.txt")
            .map(x => ( (x.split(",")(0),x.split(",")(2)) ,
(x.split(",")(3).toInt)))

baseRDD.foreach(println)

val rddMap = baseRDD.mapValues(x => (x,1))

val rddReduce = rddMap.reduceByKey((x,y)=>
(x._1+y._1,x._2+y._2));
rddReduce.foreach(println)

val StdAvg = rddReduce.mapValues{ case(sum,count) =>
(1.0*sum)/count }
StdAvg.foreach(println)
```

3. Matrix Multiplication :

Code:

```
import org.apache.spark.mllib.linalg.{Vectors, Matrices}
import org.apache.spark.mllib.linalg.distributed.{IndexedRowMatrix,
  IndexedRow}

val rows =  sc.parallelize(Seq(
  (0L, Array(1.0, 0.0, 0.0)),
  (0L, Array(0.0, 1.0, 0.0)),
  (0L, Array(0.0, 0.0, 1.0)))
).map{case (i, xs) => IndexedRow(i, Vectors.dense(xs))}

val indexedRowMatrix = new IndexedRowMatrix(rows)
val localMatrix = Matrices.dense(3, 2, Array(1.0, 2.0, 3.0, 4.0, 5.0, 6.0))

indexedRowMatrix.multiply(localMatrix).rows.collect
```

Streaming Algorithms

1. Bloom Filter:

Code:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
from bloomfilter import BloomFilter, ScalableBloomFilter, SizeGrowthRate

animals = [
    "dog",
    "cat",
    "giraffe",
    "fly",
    "mosquito",
    "horse",
    "eagle",
    "bird",
    "bison",
    "boar",
    "butterfly",
    "ant",
    "anaconda",
    "bear",
    "chicken",
    "dolphin",
    "donkey",
    "crow",
    "crocodile",
]

other_animals = [
    "badger",
    "cow",
    "pig",
    "sheep",
    "bee",
    "wolf",
    "fox",
    "whale",
    "shark",
    "fish",
    "turkey",
```

```
    "duck",
    "dove",
    "deer",
    "elephant",
    "frog",
    "falcon",
    "goat",
    "gorilla",
    "hawk",
]


def bloom_filter_example():
    print("========== Bloom Filter Example ==========")
    bloom_filter = BloomFilter(size=1000, fp_prob=1e-6)

    # Insert items into Bloom filter
    for animal in animals:
        bloom_filter.add(animal)

    # Print several statistics of the filter
    print(
        "+ Capacity: {} item(s)".format(bloom_filter.size),
        "+ Number of inserted items: {}".format(len(bloom_filter)),
        "+ Filter size: {} bit(s)".format(bloom_filter.filter_size),
        "+ False Positive probability: {}".format(bloom_filter.fp_prob),
        "+ Number of hash functions: {}".format(bloom_filter.num_hashes),
        sep="\n",
        end="\n\n",
    )

    # Check whether an item is in the filter or not
    for animal in animals + other_animals:
        if animal in bloom_filter:
            if animal in other_animals:
                print(
                    f"{animal}" is a FALSE POSITIVE case (please adjust fp_prob to
a smaller value).'
                )
            else:
```

```
            print(f"{animal}" is PROBABLY IN the filter.')
        else:
            print(f"{animal}" is DEFINITELY NOT IN the filter as expected.')

    # Save to file
    with open("bloom_filter.bin", "wb") as fp:
        bloom_filter.save(fp)

    # Load from file
    with open("bloom_filter.bin", "rb") as fp:
        bloom_filter = BloomFilter.load(fp)


def scalable_bloom_filter_example():
    print("========== Bloom Filter Example ==========")
    scalable_bloom_filter = ScalableBloomFilter(
        initial_size=100,
        initial_fp_prob=1e-7,
        size_growth_rate=SizeGrowthRate.LARGE,
        fp_prob_rate=0.9,
    )
    # Insert items into Bloom filter
    for animal in animals:
        scalable_bloom_filter.add(animal)

    # Print several statistics of the filter
    print(
        "+ Capacity: {} item(s)".format(scalable_bloom_filter.size),
        "+ Number of inserted items: {}".format(len(scalable_bloom_filter)),
        "+ Number of Bloom filters:
{}".format(scalable_bloom_filter.num_filters),
        "+ Total size of filters: {} bit(s)".format(scalable_bloom_filter.filter_size),
        "+ False Positive probability: {}".format(scalable_bloom_filter.fp_prob),
        sep="\n",
        end="\n\n",
    )

    # Check whether an item is in the filter or not
    for animal in animals + other_animals:
        if animal in scalable_bloom_filter:
```

```
            if animal in other_animals:
                print(
                    f'"{animal}" is a FALSE POSITIVE case (please adjust fp_prob to
a smaller value).'
                )
            else:
                print(f'"{animal}" is PROBABLY IN the filter.')
        else:
            print(f'"{animal}" is DEFINITELY NOT IN the filter as expected.')

    # Save to file
    with open("scalable_bloom_filter.bin", "wb") as fp:
        scalable_bloom_filter.save(fp)

    # Load from file
    with open("scalable_bloom_filter.bin", "rb") as fp:
        scalable_bloom_filter = ScalableBloomFilter.load(fp)


if __name__ == "__main__":
    bloom_filter_example()
    scalable_bloom_filter_example()
```

2. FM Algo Code:

```
from bitarray import bitarray
import mmh3
import statistics
import math

def trailing_zeros(n):
        s = str(n)
        return len(s)-len(s.rstrip('0'))

input_file =
['quotes_2008-08.txt','quotes_2008-09.txt','quotes_2008-10.txt','quotes_2008-1
1.txt','quotes_2008-12.txt',

'quotes_2009-01.txt','quotes_2009-02.txt','quotes_2009-03.txt','quotes_2009-04
.txt']

result = [ "" for i in range(10)]
result_tail = [[] for i in range(10)]

for i in input_file:
        fp = open(i,"r", encoding='ISO-8859-1')
        for line in fp:
                stream = line.split("\t")
                if stream[0] is 'Q':
                        for seed in range(10):
                                result[seed] = format(abs(mmh3.hash(stream[1],
seed)), '032b')

result_tail[seed].append(trailing_zeros(result[seed]))
        fp.close()
group1 = (2**(max(result_tail[0])) + 2**(max(result_tail[1])) +
2**(max(result_tail[2])) + 2**(max(result_tail[3])) +
2**(max(result_tail[4])))/ float(5)
group2 = (2**(max(result_tail[5])) + 2**(max(result_tail[6])) +
2**(max(result_tail[7])) + 2**(max(result_tail[8])) +
2**(max(result_tail[9])))/ float(5)
print (math.ceil(statistics.median([group1, group2])))
```

3.DGIM Algorithm:

```
import IPython
import sys
import itertools
import time
import math

def checkAndMergeBucket(bucketList, t):
    bucketListLength = len(bucketList)
    for i in range (bucketListLength):
        if len(bucketList[i]) > 2:
            bucketList[i].pop(0)
            if i + 1 >= bucketListLength:
                bucketList[i].pop(0)
            else:
                bucketList[i+1].append(bucketList[i].pop(0))

K = 1000
N = 1000
k = int(math.floor(math.log(N, 2)))
t = 0
onesCount = 0
bucketList = []
for i in range(k+1):
    bucketList.append(list())

with open('engg5108_stream_data.txt') as f:
    while True:
        c = f.read(1)
        if not c:
            for i in range(k+1):
                for j in range(len(bucketList[i])):
                    print "Size of bucket: %d, timestamp: %d" % (pow(2,i),
bucketList[i][j])
                    earliestTimestamp = bucketList[i][j]
            for i in range(k+1):
                for j in range(len(bucketList[i])):
                    if bucketList[i][j] != earliestTimestamp:
```

```
            onesCount = onesCount + pow(2,i)
        else:
            onesCount = onesCount + 0.5 * pow(2,i)
    print "Number of ones in last %d bits: %d" % (K, onesCount)
    break
t = (t + 1) % N
for i in range(k+1):
    for bucketTimestamp in bucketList[i]:
        if bucketTimestamp == t:
            bucketList[i].remove(bucketTimestamp)
if c == '1':
    bucketList[0].append(t)
    checkAndMergeBucket(bucketList, t)
elif c == '0':
    continue
```

4.Decay window Algorithm Code:

```
# O(n) solution for finding
# maximum sum of a subarray of size k


def maxSum(arr, k):
    # length of the array
    n = len(arr)

    # n must be greater than k
    if n < k:
        print("Invalid")
        return -1

    # Compute sum of first window of size k
    window_sum = sum(arr[:k])

    # first sum available
    max_sum = window_sum

    # Compute the sums of remaining windows by
    # removing first element of previous
    # window and adding last element of
    # the current window.
    for i in range(n - k):
        window_sum = window_sum - arr[i] + arr[i + k]
        max_sum = max(window_sum, max_sum)

    return max_sum


# Driver code
arr = [1, 4, 2, 10, 2, 3, 1, 0, 20]
k = 4
print(maxSum(arr, k))
```

HIVE

Introduction: Hive is a data warehouse infrastructure tool to process structured data in Hadoop. It resides on top of Hadoop to summarise Big Data, and makes querying and analysing easy.

Experiments:

1. **Table Creation**

```
CREATE TABLE IF NOT EXISTS student(
Student_Name STRING,
Student_Rollno INT,
Student_Marks FLOAT)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';
```

2. **Add data to Hive Table**

   1. *Using INSERT command*
      INSERT INTO TABLE student VALUES (‘Anjali’,1,'95'),(‘Muskan’, 2 , '96'),('Chahat',3,'90');

   2. *Load Data Statement*

      LOAD DATA LOCAL INPATH '/home/anjali/Documents/data.csv' INTO TABLE student;

3. **Read Content from Table**

*SELECT * FROM student;*

4. **Partitioning in Hive**

CREATE TABLE student(student_name STRING ,father_name STRING ,percentage FLOAT)
partitioned by (section STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';

describe student;

PIG

Introduction: Apache Pig is a platform for analysing large data sets that consists of a high-level language for expressing data analysis programs, coupled with infrastructure for evaluating these programs. The salient property of Pig programs is that their structure is amenable to substantial parallelization, which in turns enables them to handle very large data sets.

Experiment:

1. **Load data and write Pig script**

A = LOAD 'myfile'

```
AS (x, y, z);

B = FILTER A by x > 0;

C = GROUP B BY x;

D = FOREACH A GENERATE
x, COUNT(B);

STORE D INTO 'output';
```

2. JOINs and GROUP, ORDER BY CLAUSE IN PIG

```
SELECT c_id ,
SUM(amount) AS CTotal
FROM customers c

JOIN sales s ON c.c_id = s.c_id

WHERE c.city = 'Texas'
GROUP BY c_id
HAVING SUM(amount) > 2000
ORDER BY CTotal DESC
```