Business Intelligence & Big Data Analytics Journal

M.Sc Part I Computer Science

Vedant Jadhav 524

March 22, 2023

R.J. College of Arts, Science & Commerce
Business Intelligence & Big Data Analytics
Seat number: 524

Contents

List of Figures

Practical No: 1

Aim: Installing and setting environment variables for Working with Apache Hadoop

Practical:

1. Download the latest stable release of Apache Hadoop from the official website.
2. Extract the downloaded file to a directory on your system.
3. Set the `HADOOP_HOME` environment variable to the location where Hadoop is installed using the following command: `export HADOOP_HOME=/path/to/hadoop`
4. Set the `JAVA_HOME` environment variable to the location where Java is installed using the following command: `export JAVA_HOME=/path/to/java`
5. Add the `bin` directory of Hadoop to your `PATH` environment variable using the following command: `export PATH=$PATH:$HADOOP_HOME/bin`
6. Configure the Hadoop cluster by editing the `hadoop-env.sh` file located in the `$HADOOP_HOME/etc/hadoop/` directory. Set the `JAVA_HOME` and other configuration parameters as required.
7. Configure the Hadoop core-site.xml file by setting the Hadoop filesystem URI and other properties as required. This file is located in the `$HADOOP_HOME/etc/hadoop/` directory.
8. Configure the Hadoop hdfs-site.xml file by setting the replication factor and block size for HDFS as required. This file is also located in the `$HADOOP_HOME/etc/hadoop/` directory.
9. Start the Hadoop daemons using the following command: `$HADOOP_HOME/sbin/start-all.sh`
10. Verify that Hadoop is running by accessing the web interface at `http://localhost:50070/`.

Practical No: 2

Aim: Implementing Map-Reduce Program for Word Count problem.

Description:

Here's the step-by-step procedure for implementing a Map-Reduce program for the Word Count problem:

1. Map function: The map function takes an input record (a line of text) and outputs a list of key-value pairs, where the key is a word in the line and the value is 1.

```
$\operatorname{Map}(key, value):$
    words $\gets$ Split value on whitespace
    for each word in words:
        EmitIntermediate(word, 1)
```

2. Shuffle and sort: The intermediate key-value pairs are grouped by key and sorted by key, so that all occurrences of a particular word are grouped together. This step is done automatically by the Map-Reduce framework.

3. Reduce function: The reduce function takes an intermediate key and a list of values and outputs a single key-value pair, where the key is the word and the value is the total count of that word across all input records. Here's the reduce function in LaTeX code:

```
$\operatorname{Reduce}(key, values):$
    count $\gets$ 0
    for each value in values:
        count $\gets$ count + value
    Emit(key, count)
```

4. Driver program: The driver program sets up the Map-Reduce job by specifying the input and output paths, the map and reduce functions, and any other configuration options. Here's an example of a driver program in LaTeX code:

```
$\operatorname{main}():$
    job $\gets$ new MapReduceJob()
    job.setInputPath("/path/to/input")
    job.setOutputPath("/path/to/output")
    job.setMapper(Map)
```

```
    job.setReducer(Reduce)
    job.run()
```

This is a basic outline of how a Map-Reduce program for the Word Count problem works. Note that there are many variations and optimizations of this basic approach, depending on the specifics of the input data and the desired output.

Code:

```
import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {

  public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }

```

```
  public static class IntSumReducer
       extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,
                       Context context
                       ) throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
      }
      result.set(sum);
      context.write(key, result);
    }
  }

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

Output :

The seven-segment will display the numbers starting from 0 to 9 after every 1000 ms as the delay de is declared as 1000.

Practical No: 3

Practical

Pig is a high-level scripting language used for analyzing large datasets. It provides a simple syntax for performing data transformations and analysis. Here's an example Pig script for solving counting problems: Assume we have a dataset containing the following information about customers:

```

(John, 30, M)
(Mary, 25, F)
(Tom, 40, M)
(Emily, 35, F)
```

The goal is to count the number of male and female customers in the dataset. Here's the Pig script to accomplish this task:

```

-- Load the dataset
customers = LOAD "path/to/customers.csv" USING PigStorage(",")
    AS (name: chararray, age: int, gender: chararray);

-- Group the data by gender
gender_group = GROUP customers BY gender;

-- Count the number of records in each group
gender_count = FOREACH gender_group GENERATE group, COUNT(customers);

-- Store the result
STORE gender_count INTO "path/to/output";
```

In the above script, we first load the dataset using the LOAD command and define the schema of the data using the AS keyword. Then, we group the data by the gender field using the GROUP command. Next, we use the FOREACH command to count the number of records in each group using the COUNT function. Finally, we store the result using the STORE command.
The output will be stored in a file at the specified output path, containing the following records:

```
(M, 2)
(F, 2)
```

This shows that there are 2 male and 2 female customers in the dataset.

Practical No: 4

Aim: Install HBase and use the HBase Data model Store and retrieve data.

Overview:

1. Start by creating a table in HBase, specifying its name and column families. A column family is a group of related columns that are stored together. For example, you might have a column family called "personal" for storing personal information, and another called "professional" for storing professional information.
2. To store data in HBase, you need to create a Put object for each row of data you want to insert. The Put object specifies the row key, the column family, the column qualifier (which identifies the specific column within the column family), and the value.
3. Once you have created your Put objects, you can use the HTable.put() method to insert them into HBase.
4. To retrieve data from HBase, you can create a Get object for each row of data you want to retrieve. The Get object specifies the row key, and can also specify the column family and column qualifier if you want to retrieve a specific column.
5. You can then use the HTable.get() method to retrieve the data. This method returns a Result object, which contains the data for the specified row.
6. To extract the data from the Result object, you can use the getXXX() methods, where XXX is the data type of the column. For example, if you have a column called "age" that contains an integer, youcan use the Result.getInt() method to retrieve the value.
7. You can also use scan operations to retrieve multiple rows of data at once. A scan operation returns a ResultScanner object, which allows you to iterate over the results.
8. When you are done working with your table, you should close the HTable object to free up resources.

Code:

```
//Start HBase
hbase shell
//HBase Commands status
version,
table_help
whoami
//Data  Definition  Language
create  'employee', 'Name', 'ID',
```

```
    ' Designation', ' Salary', ' Department'
//Verify created table list
//Disable single table disable 'employee' scan 'employee'
//or
is_disable 'employee'
//Disable multiple tables disable_all 'e.*'
// Enabling table enable 'employee'
//Or is_enabled"employee"
//create new table
create 'student', 'name', 'age', 'course'
put 'student', 'sharath', 'name:fullname', 'sharathkumar'
put 'student', 'sharath', 'age:presentage', '24'
put 'student', 'sharath', 'course:pursuing', 'Hadoop'
put 'student', 'shashank', 'name:fullname', 'shashank R
put 'student', 'shashank', 'age:presentage', '23'
put 'student', 'shashank', 'course:pursuing', 'Java'
//Get Information
get 'student', 'shashank'
get 'student', 'sharath'
get 'student', 'sharath', 'course' get 'student', 'shashank',
 'course' get 'student', 'sharath', 'name'
//Scan
scan 'student'
//Count
Count 'student'
//Alter
alter 'student', NAME=>' name', VERSIONS=>5
put 'student', 'shashank', 'name:fullname', 'shashank Rao'
scan 'student'
//Delete
delete 'student', 'shashank', 'name:fullname'
```

Output :

LCD screen display the message "Hello world".

Practical No: 5

Aim: : Install Hive and use Hive Create and store structured databases.

Code:

```
cat > /home/cloudera/employee.txt
1~Sachine~Pune~Product
Engineering~100000~Big Data
2~Gaurav~Banglore~Sales~90000~CRM
3~Manish~Chennai~Recruiter~125000~HR
4~Bhushan~Hyderabad~Developer~50000~BFSI
cat /home/cloudera/employee.txt
sudo -u hdfs hadoop fs -put /home/cloudera/employee.txt /inputdir hdfs dfs -ls /
hdfs dfs -ls /inputdirectory
hadoop fs -cat /inputdirectory/employee.txt
hive
show databases;
create database organization;
show databases;
use organization;
show tables;
hive> create table employee(
> id int,
> name string,
> city string,
> department string,
> salary int,
> domain string)
> row format delimited
> fields terminated by "~";
show tables;
select * from employee;
show tables;
load data inpath "/inputdir/employee.txt" overwrite into table employee;
show tables;
select * from employee;
```

Output :

Practical No: 6

Aim:Write a program to construct different types of k-shingles for a given document.

Overall:

1. Install and configure Hive: First, you need to install and configure Hive on your system. You can follow the instructions provided in the Hive documentation for your specific environment to do this.
2. Define your data schema: Next, you need to define the schema for your data. This involves specifying the data types and column names for each field in your dataset. You can define your schema using the CREATE TABLE statement in Hive.
3. Create a Hive table: Once you have defined your schema, you can create a Hive table to store your data. You can do this using the CREATE TABLE statement in Hive, and specifying the table name and the schema you defined in the previous step.
4. Load data into your table: Once your table is created, you can load data into it. You can do this using the LOAD DATA statement in Hive, which allows you to specify the source of your data and the destination table.
5. Query your table: With your data loaded into your table, you can now query it using the SELECT statement in Hive. This allows you to retrieve specific subsets of your data based on criteria such as date ranges, locations, or other filters.
6. Analyze your data: Finally, with your data stored in Hive, you can use Hive's built-in data analysis functions to gain insights into your data. For example, you can use Hive's GROUP BY and ORDER BY clauses to group and sort your data, or use Hive's JOIN statement to combine data from multiple tables.

Code:

```
install.packages("tm")
require("tm")
install.packages("devtools")
readinteger <- function () {
  n <-readline(prompt = "Enter value of k-1: ")
  k <-as.integer(n)
  u1 <-readLines("c:/msc/r-corpus/File1.txt")
  Shingle <-0
  i <-0
  while (i < nchar(u1) - k + 1) {
```

```
    Shingle[i] <- substr(u1, start = i, stop = i + k)
    print(Shingle[i])
    i = i + 1
  }
}
if (interactive()) readinteger()
```

Output :

```
> if(interactive()) readinteger()
Enter value of k-1: 2
character(0)
[1] "thi"
[1] "his"
[1] "is "
[1] "s i"
[1] " is"
[1] "is "
[1] "s a"
[1] " a "
[1] "a t"
[1] " te"
[1] "tex"
[1] "ext"
[1] "xt."
[1] "t. "
[1] ". i"
[1] " it"
[1] "it "
[1] "t i"
[1] " is"
[1] "is "
[1] "s s"
[1] " sh"
[1] "sho"
[1] "hor"
[1] "ort"
[1] "rt,"
[1] "t, "
[1] ", a"
[1] " an"
```

```
[1] "and"
[1] "nd "
[1] "d i"
[1] " it"
[1] "it "
[1] "t i"
[1] " is"
[1] "is "
[1] "s o"
[1] " on"
[1] "one"
[1] "ne "
[1] "e l"
[1] " li"
[1] "lin"
[1] "ine"
[1] "ne."
[1] "e. "
[1] ". i"
[1] " it"
[1] "it "
[1] "t i"
....

```

Practical No: 7

Aim: Write a program for measuring similarity among documents and detecting passages which have been reused.

Description:

1. Install and load required packages: `tm`, `ggplot2`, `textreuse`, and `devtools`.
2. Create a Corpus object `my.corpus` from a directory of text files using `Corpus(DirSource())`.
3. Remove English stopwords from `my.corpus` using `tm_map()`.
4. Create a TermDocumentMatrix object `my.tdm` from `my.corpus` using `TermDocumentMatrix()`.
5. Create a DocumentTermMatrix object `my.dtm` from `my.corpus` using `DocumentTermMatrix()` with term frequency-inverse document frequency weighting and stopword removal.
6. Convert `my.tdm` to a data frame `my.df` using `as.data.frame(inspect(my.tdm))`.
7. Scale `my.df` using `scale()`.
8. Compute the Euclidean distance matrix `d` from the scaled data using `dist()`.
9. Cluster the documents using hierarchical clustering with Ward's linkage method using `hclust()`.
10. Plot the dendrogram of the hierarchical clustering using `plot()`.

Code:

```
install.packages("tm")
require("tm")
install.packages("ggplot2")
install.packages("textreuse")
install.packages("devtools")
my.corpus <- Corpus(DirSource("C:/MSC Notes/r-corpus"))
my.corpus <- tm_map(my.corpus, removeWords, stopwords("english"))
my.tdm <- TermDocumentMatrix(my.corpus)
my.dtm <- DocumentTermMatrix(my.corpus,
    control = list(weighting = weightTfIdf, stopwords = TRUE))
my.df <- as.data.frame(inspect(my.tdm))
my.df.scale <- scale(my.df)
d <- dist(my.df.scale, method = "euclidean")
fit <- hclust(d, method = "ward")
plot(fit)
```

Output :

```
<TermDocumentMatrix (terms: 69, documents: 6)>>
Non-/sparse entries: 97/317
Sparsity           : 77%
Maximal term length: 12
Weighting          : term frequency (tf)
Docs
.
...
```

Practical No: 8

Aim: Write a program to compute the n- moment for a given stream where n is given.

Overall

:

Here are the steps to write a program to compute the N-moment for a given stream:

1. Define the input: The input to the program is a stream of numbers and the value of N, which represents the order of the moment that needs to be computed.
2. Define the variables: Define the variables needed to compute the moment. We need a variable to store the current sum, a variable to store the number of items in the stream, and a variable to store the current value of the moment.
3. Read the input stream: Read the input stream one value at a time. For each value, add it to the current sum, increment the count of items, and calculate the current value of the moment using the formula: $moment = sum(item^N)/count$.
4. Output the result: Once all the values in the stream have been processed, output the final value of the moment.

Code:

```
package pract;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.stream.Stream;

public class Pract {

  public static void main(String[] args) {
    int n = 15;

    String stream[] = {
      "a",
      "b",
      "c",
      "b",
      "d",
```

```
        "a",
        "c",
        "d",
        "a",
        "b",
        "d",
        "c",
        "a",
        "a",
        "b"
    };


    int zeroMoment = 0, firstMoment = 0, secondMoment = 0, count = 1, flag = 0;

    ArrayList < Integer > arrlist = new ArrayList();

    System.out.println("ArrayList element are:");

    for (int i = 0; i < 5; i++) {
      System.out.println(stream[i] + " ");
    }
    Arrays.sort(stream);

    for (int i = 1; i < n; i++) {
      if (stream[i] == stream[i - 1]) {
        count++;
      } else {
        arrlist.add(count);
        count = 1;
      }
    }
    arrlist.add(count);

    zeroMoment = arrlist.size();

    System.out.println("Value of zero moment is:" + zeroMoment);

    for (int i = 0; i < arrlist.size(); i++) {
      firstMoment += arrlist.get(i);
    }
    System.out.println("Values of firstMoment:" + firstMoment);
```

```
59
60      for (int i = 0; i < arrlist.size(); i++) {
61        int j = arrlist.get(i);
62        secondMoment += (j * j);
63      }
64      System.out.println("value of secont moment is:" + secondMoment);
65    }
66
67  }
```

Output :

When the program is loaded in arduino UNO, Servo motor starts rotating.

Practical No: 9

Aim: Write a program to demonstrate the Alon-Matias-Szegedy Algorithm for second moments.

Overview

The Alon-Matias-Szegedy (AMS) algorithm is a randomized algorithm that estimates the second moments of a dataset. It is commonly used in data stream algorithms and is useful for analyzing large datasets that cannot be stored in memory. The basic idea behind the algorithm is to hash the data in a way that preserves the second moments and then estimate the second moments from the hashed values. Here are the steps to implement the AMS algorithm:

1. Step 1: Hash the data Choose a family of pairwise-independent hash functions, $h : U \rightarrow 1, 1$, whereU is the universe of the input dataset. Apply each hash function h to each element of the dataset to obtain a hashed value. Store the hashed values in a table.
2. Step 2: Compute the first moment Compute the sum of the hashed values. This gives an estimate of the first moment of the dataset.
3. Step 3: Compute the second moment For each element in the dataset, compute the sum of the hashed values for all pairs of elements that contain that element. Compute the average of the sum of the hashed values for each element. This gives an estimate of the second moment of the dataset.
4. Step 4: Return the estimate Return the estimate of the second moment obtained in step 3. Note that the AMS algorithm may not give an exact estimate of the second moment, but it has been shown to provide a good approximation with high probability. The accuracy of the estimate depends on the number of hash functions used and the size of the dataset. By using more hash functions or increasing the size of the table, the accuracy of the estimate can be improved.

Code:

```java
import java.io.*;
import java.util.*;
class AMSA {
  public static int findCharCount(String stream, char XE, int random, int n) {
    int countoccurance = 0;
    for (int i = random; i < n; i++) {
      if (stream.charAt(i) == XE) {
        countoccurance++;
      }
    }
    return countoccurance;
```

```
  }
  public static int estimateValue(int XV1, int n) {
    int ExpValue;
    ExpValue = n * (2 * XV1 - 1);
    return ExpValue;
  }
  public static void main(String args[]) {
    int n = 15;
    String stream = "abcbdacdabdcaab";
    int random1 = 3, random2 = 8, random3 = 13;
    char XE1, XE2, XE3;
    int XV1, XV2, XV3;
    int ExpValuXE1, ExpValuXE2, ExpValuXE3;
    int apprSecondMomentValue;
    XE1 = stream.charAt(random1 - 1);
    XE2 = stream.charAt(random2 - 1);
    XE3 = stream.charAt(random3 - 1);
    XV1 = findCharCount(stream, XE1, random1 - 1, n);
    XV2 = findCharCount(stream, XE2, random2 - 1, n);
    XV3 = findCharCount(stream, XE3, random3 - 1, n);
    System.out.println(XE1 + "=" + XV1 + " " + XE2 + "=" + XV2 + " " + XE3 + "=" + XV3);
    ExpValuXE1 = estimateValue(XV1, n);
    ExpValuXE2 = estimateValue(XV2, n);
    ExpValuXE3 = estimateValue(XV3, n);
    System.out.println("Expected value for" + XE1 + " is::" + ExpValuXE1);
    System.out.println("Expected value for" + XE2 + " is::" + ExpValuXE2);
    System.out.println("Expected value for" + XE3 + " is::" + ExpValuXE3);
    apprSecondMomentValue = (ExpValuXE1 + ExpValuXE2 + ExpValuXE3) / 3;
    System.out.println("approximate second moment value using alon-matis-szegedy is::"
        + apprSecondMomentValue);
  }
}
```

Output:

```
  mithunparab @Nightfury16 ~/Downloads/MSc/BIG D % java prac9.java
c=3 d=2 a=2
Expected value forc is::75
Expected value ford is::45
Expected value fora is::45
```

```
approximate second moment value using alon-matis-szegedy is::55
```