PRACTICAL 01: Generate Regression Model and interpret the Result for a given Data Set.

Step-01: Open Weka-Tool > Explorer > *Preprocess* tab> Open File > *Ionosphere*.arff.

Step-02: Click *Classify* tab> Click *Choose* > *Weka >* functions > *LinearRegression*

Step-03: Select any of the **Parameters in the Dropdown-List** on the Left and then click **Start** to apply Regression on the Input File. Here: In this Example, I have selected *(Num) a07* as a Parameter.

PRACTICAL 02: Generate Forecasting Model and interpret the Result for a given Data Set.

Step-01: Open Weka-Tool > Explorer > ***Preprocess*** tab> Open File > *Weather*.arff.

Step-02: Click ***Cluster*** tab> Click *Choose* > *Weka >* Clusterers > *SimpleKMeans*

Step-03: Click **Start** to apply Regression on the Input File.

PRACTICAL 03: Write a Map-Reduce Program to count the Number of Occurrences of each alphabetic Character in the given Dataset. The Count for each Letter should be case-sensitive include both upper-case and lower-case versions of the Letter; ignore on-alphabetic Characters.

1. Use the Instructions provided at

<https://github.com/AmolGhdgnkr/Shiv/blob/master/HADOOP_INSTALLATION_WINDOWS.txt>,

and download the pre-requisite Files provided at

<https://github.com/AmolGhdgnkr/Shiv/blob/master/HADOOP_Custom_bin_Files_For_Windows.rar>

to install **Hadoop**.

1. **CharCount.java**

import java.io.IOException;  
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.lib.input.FileInputFormat;  
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;  
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;  
import java.util.StringTokenizer;  
import org.apache.hadoop.io.LongWritable;  
import org.apache.hadoop.mapreduce.Mapper;  
import org.apache.hadoop.mapreduce.Reducer;  
public class CharCount {  
    public static void main(String[] args) throws Exception {  
        Configuration conf = new Configuration();  
        Job job = new Job(conf, "CharCount");  
        job.setJarByClass(CharCount.class);  
        job.setMapperClass(Charmap.class);  
        job.setReducerClass(Charreduce.class);  
        job.setInputFormatClass(TextInputFormat.class);  
        job.setOutputFormatClass(TextOutputFormat.class);  
        job.setMapOutputKeyClass(Text.class);  
        job.setMapOutputValueClass(IntWritable.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);    }}  
class Charmap extends Mapper<LongWritable, Text, Text, IntWritable> {  
    public void map(LongWritable key, Text value, Context context)  
            throws IOException, InterruptedException {

String line = value.toString();  
        char[] carr = line.toCharArray();  
        for (char c : carr) {  
            System.out.println(c);  
            context.write(new Text(String.valueOf(c)), new IntWritable(1));        }    }}  
class Charreduce extends Reducer<Text, IntWritable, Text, IntWritable> {  
    public void reduce(Text key,Iterable<IntWritable>values,Context context)throws IOException,InterruptedException{  
        int count = 0;  
        IntWritable result = new IntWritable();  
        for (IntWritableval : values) {  
            count +=val.get();  
            result.set(count);        }  
        String found = key.toString();  
        if (found.equals("a") || found.equals("t") || found.equals("c") || found.equals("g")) {  
            context.write(key, result);            }        } }

1. The following commands are to create a directory to store the compiled java classes.

Home directory is C:\hadoop-2.8.0\sbin

Save the .java file to C:\hadoop-2.8.0\sbin

Executable Code in C:\hadoop-2.8.0\sbin

1. In the Command Prompt Window: run the following Commands successively.
2. **hdfs namenode -format**

then enter: N

1. **start-all**
2. To verify: In Browser, run: <http://localhost:8088/>
3. Get the back to Command Prompt and run the following Commands.
4. mkdir wordcountdata
5. Creating wordcountdata in Hadoop filesystem
6. C:\hadoop-2.8.0\sbin>hdfs dfs -mkdir /wordcountdata
7. Compiling the java code:
8. C:\hadoop-2.8.0\sbin>javac -classpath hadoop-core-1.2.1.jar -d wordcountdata WordCount.java
9. Creating jar file
10. C:\hadoop-2.8.0\sbin>jar -cvf MRProgramsDemo.jar -C wordcountdata/ .
11. Shifting wordcountfile.txt from Hadoop home folder to Hadoop file system
12. C:\hadoop-2.8.0\sbin>/hadoop-2.8.0/bin/hadoop fs -put /hadoop-2.8.0/wordCountFile.txt wordCountFile
13. Executing the program
14. C:\hadoop-2.8.0\sbin>/hadoop-2.8.0/bin/hadoop jar MRProgramsDemo.jar WordCount wordCountFile MRDir1
15. Checking if output is created
16. C:\hadoop-2.8.0\sbin>/hadoop-2.8.0/bin/hadoop fs -ls MRDir1
17. Seeing the output:
18. C:\hadoop-2.8.0\sbin>/hadoop-2.8.0/bin/hadoop fs -cat MRDir1/part-r-00000
19. TO DELETE DIRECTORY:
20. C:\hadoop-2.8.0\sbin>hdfs dfs -rm -r MRDir1
21. Deleted MRDir1

PRACTICAL 04: Write a Map-Reduce Program to count the Number of Occurrences of each Word in the given Dataset.

1. Use the Instructions provided at

<https://github.com/AmolGhdgnkr/Shiv/blob/master/HADOOP_INSTALLATION_WINDOWS.txt>,

and download the pre-requisite Files provided at

<https://github.com/AmolGhdgnkr/Shiv/blob/master/HADOOP_Custom_bin_Files_For_Windows.rar>

to install **Hadoop**.

1. To verify: In Browser, run: <http://localhost:8088/>
2. **WordCount.java**

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.clas

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); }}

1. The following commands are to create a directory to store the compiled java classes.

Home directory is C:\hadoop-2.8.0\sbin

Save the .java file to C:\hadoop-2.8.0\sbin

Executable Code in C:\hadoop-2.8.0\sbin

1. In the Command Prompt Window: Commands are run in the following Order at **C:\** Prompt:

C:\ > start-dfs.cmd

C:\ > start-yarn.cmd

C:\ > hdfs dfsadmin -safemode leave

C:\ > hadoop fs -mkdir /*MyInputDir*

C:\ > hadoop fs -put C:/*MyInputFile*.txt /*MyInputDir*

C:\ > hadoop fs -ls /*MyInputDir*/

C:\ > hdfs dfs -cat /*MyInputDir*/*MyInputFile*.txt

C:\ > hadoop jar C:/*MapReduceClient*.jar wordcount /*MyInputDir* /*MyOutputDir*

C:\ > hdfs dfs -cat /*MyOutputDir*/\*

C:\ > hadoop fs -rm -r /*MyInputDir*/*MyInputFile*.txt

C:\ > hadoop fs -rm -r /*MyInputDir*

PRACTICAL 05: Write a Program to construct different Types of k-Shingles for a given Document.

1. Start r-Console.
2. Install the Packages: **devtools, tm**. To do that, go to Menu: Packages > Install Package(s)…> Secure CRAN Mirror > Select a Mirror-Site > Packages > Select a Package > OK
3. Program:

readinteger <- function()

{

n <- readline(prompt=" Enter Value for (k-1): ")

k<-as.integer(n)

ul<-readLines("S:/r-corpus/MyFile.txt")

Shingle<-0

i<-0

while(i<nchar(ul)-k+1){

Shingle[i] <- substr(ul, start=i, stop=i+k)

print(Shingle[i])

i=i+1

}

}

if(interactive()) readinteger()

PRACTICAL 06: Write a Program to measure Similarity among Documents and detecting Passages which have been reused.

1. Start r-Console.
2. Install the Packages: **textreuse**, **ggplot2**, **devtools, tm**. To do that, go to Menu: Packages > Install Package(s)…> Secure CRAN Mirror > Select a Mirror-Site > Packages > Select a Package > OK
3. Select **Yes** when it requests to create a Local Library.
4. Load the packages also into your session via

**library(tm)**

1. Program:

my.corpus <- Corpus(DirSource("S:/r-corpus"))

my.corpus <- tm\_map(my.corpus, removeWords, stopwords("english"))

my.tdm <- TermDocumentMatrix(my.corpus)

#inspect(my.tdm)

my.dtm <- DocumentTermMatrix(my.corpus, control = list(weighting = weightTfIdf, stopwords = TRUE))

#inspect(my.dtm)

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.D2")

plot(fit)

barplot(as.matrix(my.tdm))

color<-c("blue","red","green","yellow","pink","orange")

barplot(as.matrix(my.tdm),col = color)

PRACTICAL 07: Write a Program to compute the n-Moment for a given Stream where n is given.

1. Start r-Console.
2. Install the Packages: **textreuse**, **ggplot2**, **devtools, tm**.
3. n\_moment.java:

import java.io.\*;

import java.util.\*;

class n\_moment

{

public static void main(String args[])

{

int n=15;

String stream[]={"A ","m ","o ","l ","G ","h ","a ","d ","i ","g ","a ","o ","n ","k ","a ","r "};

int zero\_moment=0,first\_moment=0,second\_moment=0,count=1,flag=0;

ArrayList<Integer>arrlist=new ArrayList();;

System.out.println("Array list elements are::");

for(int i=0;i<=15;i++)

{

System.out.print(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);

zero\_moment=arrlist.size();

System.out.println("\n\n\nValue of Zeroth moment for given stream::"+zero\_moment);

for(int i=0;i<arrlist.size();i++)

{

first\_moment+=arrlist.get(i);

}

System.out.println("\n\n\nValue of First moment for given stream::"+first\_moment);

for(int i=0;i<arrlist.size();i++)

{

int j=arrlist.get(i);

second\_moment+=(j\*j);

}

System.out.println("\n\n\nValue of Second moment for given stream::"+second\_moment);

}

}

Output by commands: javac and java

PRACTICAL 08: Write a Program to compute the Alon-Matias-Szegedy Algorithm for second moment.

1. AMSA.java:

import java.io.\*;

import java.util.\*;

public 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="Amol Ghadigaonkar";

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("This Practical was performed by Amol Ghadigaonkar.");

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-Matias-Szegedy: "+apprSecondMomentValue);

}

}

Output by commands: javac and java