

# Parallel and Distributed Computing

CSE 4001

## Final Project Report

***Topic:*** Customer Behavior Prediction Using Hadoop

**Submitted To:** Prof. Srimathi C

### Group Members:

Nair Anirudha R	15BCB0074
Kush Dilip Acharya	15BCB0086
G. Hema Vishal	15BCB0094
Sheril Philip	15BCB0120



**VIT**<sup>®</sup>  
**UNIVERSITY**  
(Estd. u/s 3 of UGC Act 1956)

Vellore-632 014, Tamil Nadu, India.  
[www.vit.ac.in](http://www.vit.ac.in)

**Abstract:** In today's era of internet and technology a major fraction of the population depends on online platforms to fulfill all their needs. Grocery, garments and other amenities are mostly purchased online. To maximize the profit, satisfying the customer is of utmost importance. Thus the online shopping sites have to figure out what is trending amongst the end users and also to predict the future trends. In this project we will be focusing on the same. On behalf of the e-commerce website we will be analyzing the data provided by our clients and provide them with factors like returning users allow site owners to make appropriate changes and give the customer exactly what is needed. This allows for more customer acquisition and thus more profitability for the client.

## **I. Introduction:**

### *Naïve bayes:*

It is one of the many well-known machine learning algorithms. It is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set Naive Bayes is a classification algorithm for binary (two-class) and multi-class classification problems. The technique is easiest to understand when described using binary or categorical input values.

It is called naive bayes because the calculation of the probabilities for each hypothesis are simplified to make their calculation tractable. Rather than attempting to calculate the values of each attribute value  $P(d_1, d_2, d_3|h)$ , they are assumed to be conditionally independent given the target value and calculated as  $P(d_1|h) * P(d_2|h)$  and so on.

This is a very strong assumption that is most unlikely in real data, i.e. that the attributes do not interact. Nevertheless, the approach performs surprisingly well on data where this assumption does not hold. For some types of probability models, naive Bayes classifiers can be trained very efficiently in a supervised learning setting. In many practical applications, parameter estimation for naive Bayes models uses the method of maximum likelihood; in other words, one can work with the naive Bayes model without accepting Bayesian probability or using any Bayesian methods. Despite their naive design and apparently oversimplified assumptions, naive Bayes classifiers have worked quite well in many complex real-world situations. In 2004, an analysis of the Bayesian classification problem showed that there are sound theoretical reasons for the apparently implausible efficacy of naive Bayes classifiers.

$$P(H | E) = \frac{P(E | H) * P(H)}{P(E)}$$

where

- $P(H)$  is the probability of hypothesis  $H$  being true. This is known as the prior probability.
- $P(E)$  is the probability of the evidence (regardless of the hypothesis).

- $P(E|H)$  is the probability of the evidence given that hypothesis is true.
- $P(H|E)$  is the probability of the hypothesis given that the evidence is there.

Probabilistic model:

Using Bayes' theorem, the conditional probability can be decomposed as

$$p(C_k | \mathbf{x}) = \frac{p(C_k) p(\mathbf{x} | C_k)}{p(\mathbf{x})}$$

using Bayesian probability terminology, the above equation can be written as

$$\text{posterior} = \frac{\text{prior} \times \text{likelihood}}{\text{evidence}}$$

using the chain rule for repeated applications of the definition of conditional probability:

$$\begin{aligned} p(C_k, x_1, \dots, x_n) &= p(x_1, \dots, x_n, C_k) \\ &= p(x_1 | x_2, \dots, x_n, C_k) p(x_2, \dots, x_n, C_k) \\ &= p(x_1 | x_2, \dots, x_n, C_k) p(x_2 | x_3, \dots, x_n, C_k) p(x_3, \dots, x_n, C_k) \\ &= \dots \\ &= p(x_1 | x_2, \dots, x_n, C_k) p(x_2 | x_3, \dots, x_n, C_k) \dots p(x_{n-1} | x_n, C_k) p(x_n | C_k) p(C_k) \end{aligned}$$

the joint model can be expressed as

$$\begin{aligned} p(C_k | x_1, \dots, x_n) &\propto p(C_k, x_1, \dots, x_n) \\ &\propto p(C_k) p(x_1 | C_k) p(x_2 | C_k) p(x_3 | C_k) \dots \\ &\propto p(C_k) \prod_{i=1}^n p(x_i | C_k). \end{aligned}$$

This means that under the above independence assumptions, the conditional distribution

over the class variable is:

$$p(C_k | x_1, \dots, x_n) = \frac{1}{Z} p(C_k) \prod_{i=1}^n p(x_i | C_k)$$

**Advantages:**

Easy to implement

Very efficient

Good results obtained

***Disadvantages:***

Assumption: class conditional independence therefore loss of accuracy when the assumption is seriously violated

***Hadoop:***

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.

A Hadoop frame-worked application works in an environment that provides distributed storage and computation across clusters of computers. Hadoop is designed to scale up from single server to thousands of machines, each offering local computation and storage.

Hadoop framework includes following four modules:

- **Hadoop Common:**

These are Java libraries and utilities required by other Hadoop modules. These libraries provides filesystem and OS level abstractions and contains the necessary Java files and scripts required to start Hadoop.

- **Hadoop YARN:**

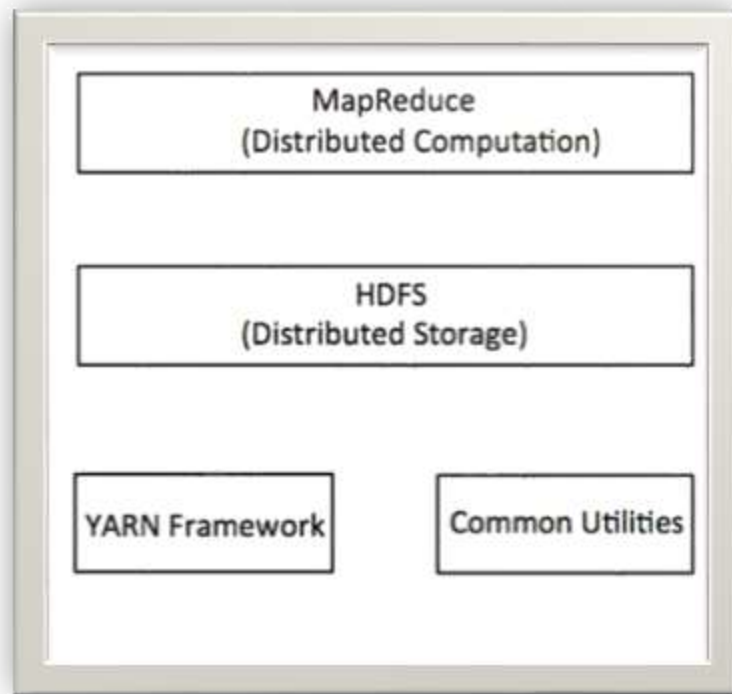
This is a framework for job scheduling and cluster resource management.

- **Hadoop Distributed File System (HDFS™):**

A distributed file system that provides high-throughput access to application data.

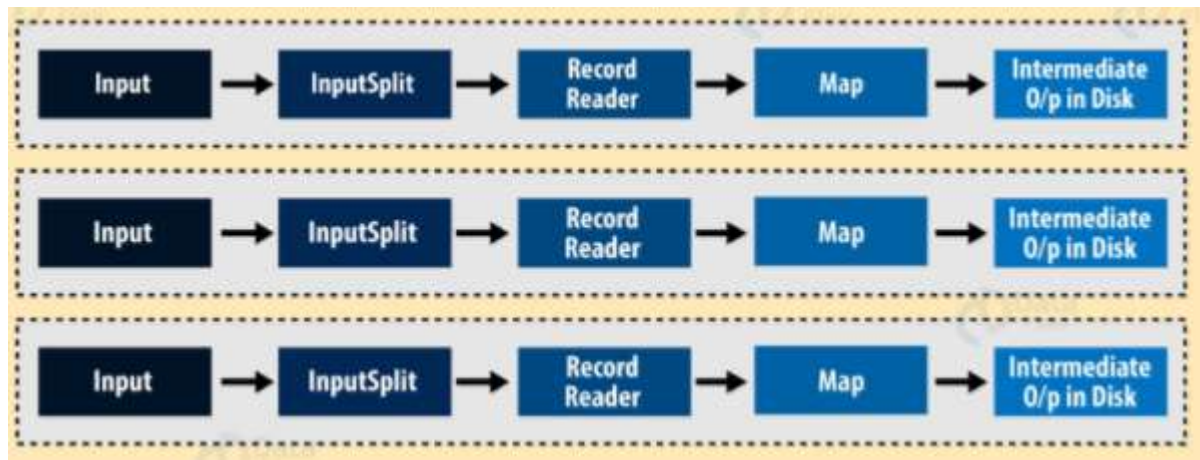
- **Hadoop MapReduce:**

This is YARN-based system for parallel processing of large data sets.



### Mapper:

Mapper task is the first phase of processing that processes each input record (from RecordReader) and generates an intermediate key-value pair. Hadoop Mapper store intermediate-output on the local disk.



Mapper task processes each input record and it generates a new <key, value> pairs. The <key, value> pairs can be completely different from the input pair. In mapper task, the output is the full collection of all these <key, value> pairs. Before writing the output for each mapper task, partitioning of output take place on the basis of the key and then sorting is done. This partitioning specifies that all the values for each key are grouped together.

MapReduce frame generates one map task for each InputSplit (we will discuss it below.) generated by the InputFormat for the job.

Mapper only understands <key, value> pairs of data, so before passing data to the mapper, data should be first converted into <key, value> pairs.

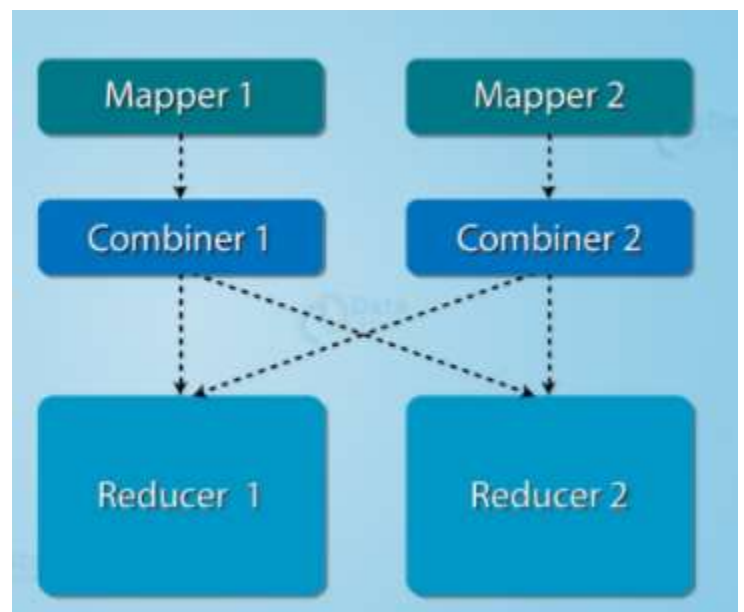
*Key value pair generated by mapper:*

- InputSplit – It is the logical representation of data. It describes a unit of work that contains a single map task in a MapReduce program.
- RecordReader – It communicates with the InputSplit and it converts the data into key-value pairs suitable for reading by the Mapper. By default, it uses TextInputFormat for converting data into the key-value pair. RecordReader communicates with the InputSplit until the file reading is not completed.

## Combiner:

Hadoop Combiner is also known as “Mini-Reducer” that summarizes the Mapper output record with the same Key before passing to the reducer.

On a large dataset when we run MapReduce job, so large chunks of intermediate data is generated by the Mapper and this intermediate data is passed on the Reducer for further processing, which leads to enormous network congestion. MapReduce framework provides a function known as Combiner that plays a key role in reducing network congestion. The combiner in MapReduce is also known as ‘Mini-reducer’. The primary job of Combiner is to process the output data from the Mapper, before passing it to Reducer. It runs after the mapper and before the Reducer and its use is optional.



### *Advantages of Combiner in MapReduce*

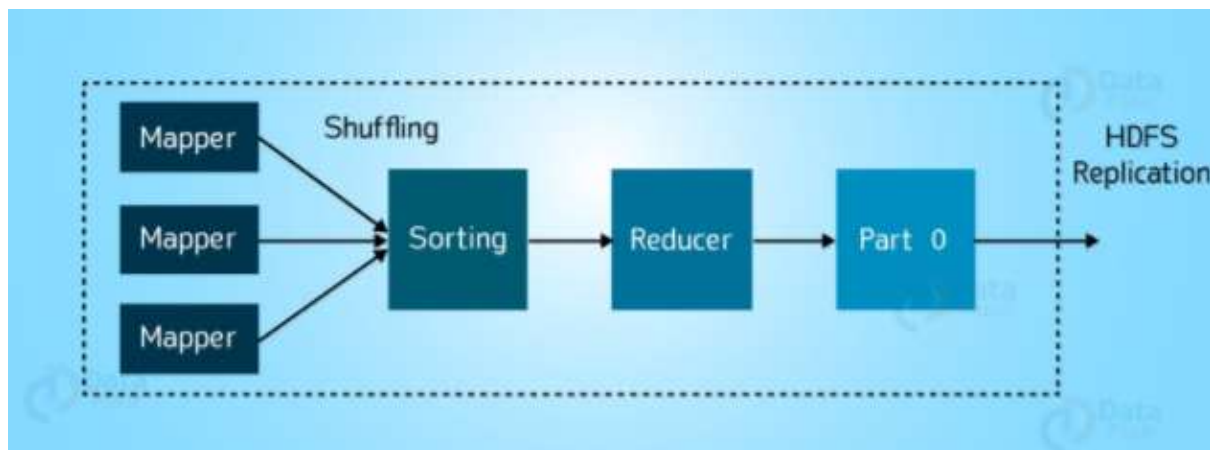
- Combiner reduces the time taken for data transfer between mapper and reducer.
- It decreases the amount of data that needed to be processed by the reducer.
- The Combiner improves the overall performance of the reducer.

### *Disadvantages of Combiner in MapReduce:*

- MapReduce jobs cannot depend on the Hadoop combiner execution because there is no guarantee in its execution.
- In the local filesystem, the key-value pairs are stored in the Hadoop and run the combiner later which will cause expensive disk IO.

### Reducer:

Reducer takes the output of the Mapper (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.



The Reducer process the output of the mapper. After processing the data, it produces a new set of output. At last HDFS stores this output data.

Reducer takes the output of the Mapper (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 Reducer takes a set of an intermediate key-value pair produced by the mapper as the input and runs a Reducer function on each of them. One can aggregate, filter, and combine this data (key, value) in several ways for a wide range of processing. Reducer first processes the intermediate values for key generated by the map function and then generates the output (zero or more key-value pair).

One-one mapping takes place between keys and reducers. Reducers run in parallel since they are independent of one another. The user decides the number of reducers. By default, number of reducers is 1.

### Phases of Reducer:

#### *Shuffle Phase*

In this phase, the sorted output from the mapper is the input to the Reducer. In Shuffle phase, with the help of HTTP, the framework fetches the relevant partition of the output of all the mappers.

#### *Sort Phase*

In this phase, the input from different mappers is again sorted based on the similar keys in different Mappers. The shuffle and sort phases occur concurrently.

#### *Reduce phase:*

In this phase, after shuffling and sorting, reduce task aggregates the key-value pairs. The `OutputCollector.collect()` method, writes the output of the reduce task to the Filesystem. Reducer output is not sorted.

#### *Mapreduce:*

MapReduce is a processing technique and a program model for distributed computing based on java. The MapReduce algorithm contains two important tasks, namely Map and Reduce. Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs). Secondly, reduce task, which takes the output from a map as an input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce task is always performed after the map job.

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.

- Generally MapReduce paradigm is based on sending the computer to where the data resides!
- MapReduce program executes in three stages, namely map stage, shuffle stage, and reduce stage.
  - **Map stage:** The map or mapper's job is to process the input data. Generally the input data is in the form of file or directory and is stored in the Hadoop file system (HDFS). The input file is passed to the mapper function line by line. The mapper processes the data and creates several small chunks of data.
  - **Reduce stage:** This stage is the combination of the **Shuffle** stage and the **Reduce** stage. The Reducer's job is to process the data that comes from the mapper. After processing, it produces a new set of output, which will be stored in the HDFS.
- During a MapReduce job, Hadoop sends the Map and Reduce tasks to the appropriate servers in the cluster.
- The framework manages all the details of data-passing such as issuing tasks, verifying task completion, and copying data around the cluster between the nodes.
- Most of the computing takes place on nodes with data on local disks that reduces the network traffic.
- After completion of the given tasks, the cluster collects and reduces the data to form an appropriate result, and sends it back to the Hadoop server.

#### *Terminology:*

- **PayLoad** - Applications implement the Map and the Reduce functions, and form the core of the job.
- **Mapper** - Mapper maps the input key/value pairs to a set of intermediate key/value pair.
- **NameNode** - Node that manages the Hadoop Distributed File System (HDFS).
- **DataNode** - Node where data is presented in advance before any processing takes place.
- **MasterNode** - Node where JobTracker runs and which accepts job requests from clients.

- **SlaveNode** - Node where Map and Reduce program runs.
- **JobTracker** - Schedules jobs and tracks the assign jobs to Task tracker.
- **Task Tracker** - Tracks the task and reports status to JobTracker.
- **Job** - A program is an execution of a Mapper and Reducer across a dataset.
- **Task** - An execution of a Mapper or a Reducer on a slice of data.
- **Task Attempt** - A particular instance of an attempt to execute a task on a SlaveNode.

### *Inputs and outputs:*

The MapReduce framework operates on  $\langle \text{key}, \text{value} \rangle$  pairs, that is, the framework views the input to the job as a set of  $\langle \text{key}, \text{value} \rangle$  pairs and produces a set of  $\langle \text{key}, \text{value} \rangle$  pairs as the output of the job, conceivably of different types.

The key and the value classes should be in serialized manner by the framework and hence, need to implement the Writable interface. Additionally, the key classes have to implement the Writable-Comparable interface to facilitate sorting by the framework. Input and Output types of a MapReduce job: (Input)  $\langle k1, v1 \rangle \rightarrow \text{map} \rightarrow \langle k2, v2 \rangle \rightarrow \text{reduce} \rightarrow \langle k3, v3 \rangle$  (Output)

	Input	Output
Map	$\langle k1, v1 \rangle$	list ( $\langle k2, v2 \rangle$ )
Reduce	$\langle k2, \text{list}(v2) \rangle$	list ( $\langle k3, v3 \rangle$ )

## **II. Architecture:**

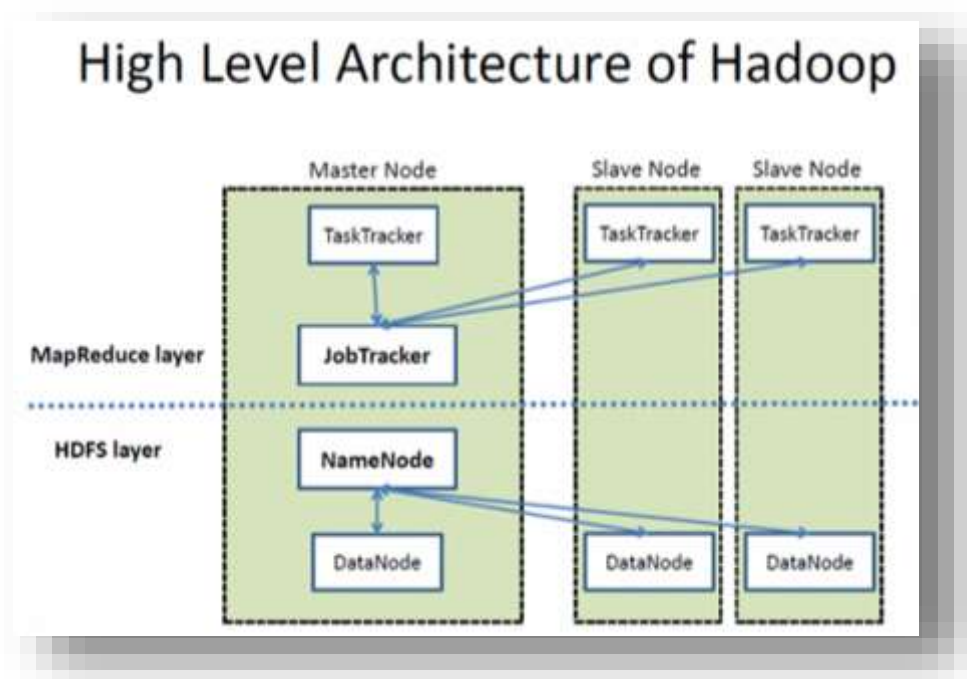
### Hadoop Architecture Overview

Apache Hadoop gives an adaptable, flexible, and reliable Big Data environment for an arrangement of frameworks with storage limit and local computing capacity using basic hardware. Hadoop takes after a Master Slave architecture for transforming and investigating vast datasets utilizing the Hadoop MapReduce paradigm. The 3 imperative segments of Hadoop that assume a crucial part in the architecture of Hadoop are:

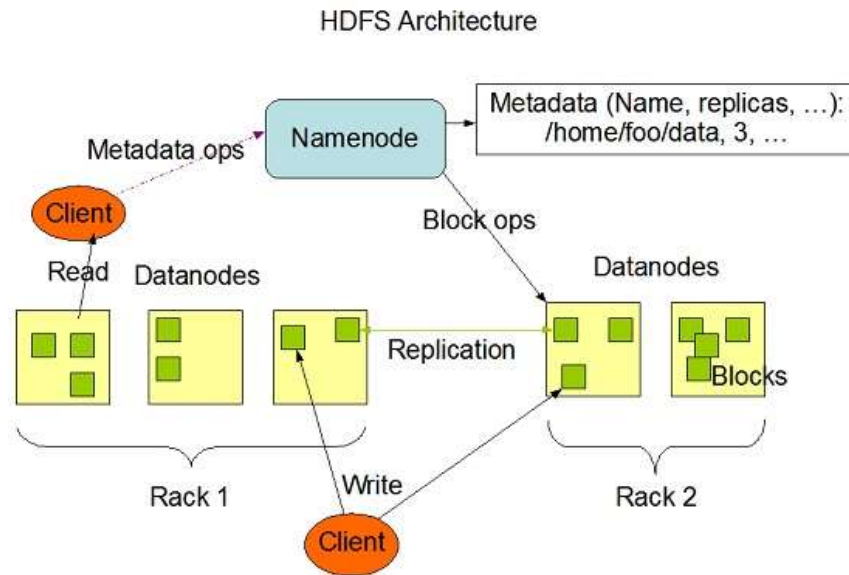
1. Hadoop Distributed File System (HDFS): UNIX File System Modeling

2. Hadoop MapReduce
3. However, another resource negotiator (YARN)

Hadoop takes after a master slave architecture outline for data storage and appropriated data preparing utilizing HDFS and MapReduce, individually. The master node for data storage is hadoop HDFS is the NameNode and the master node for parallel handling of data utilizing Hadoop MapReduce is Job Tracker. The slave nodes of the hadoop architecture are alternate machines in the Hadoop cluster that store data and perform complex figurings. Each slave node has a Task Tracker daemon and a DataNode that synchronizes forms with Job Tracker and NameNode, individually. In the design execution of Hadoop, master or slave frameworks can be arranged in the cloud or in the installations.



A file in HDFS is separated into various squares and each is reproduced in the Hadoop cluster. A piece in HDFS is a data bubble in the underlying file system with a default size of 64 MB. The extent of a piece can be extended to 256 MB as required.



The Hadoop Distributed File System (HDFS) stores application data and file system metadata independently on devoted servers. NameNode and DataNode are the two basic segments of the Hadoop HDFS architecture. Application data is put away on servers called DataNodes, and file system metadata is put away on servers named NameNode. HDFS recreates the contents of the file in various DataNodes in view of the replication factor to guarantee data unwavering quality. NameNode and DataNode speak with each other utilizing TCP conventions. For Hadoop architecture to perform well, HDFS must meet certain requirements:

- All hard drives must have elite.
- Good system speed to deal with moderate data transfer and block replication.

## NameNode

All files and registries in the HDFS namespace are represented in NameNode by nodes that contain different traits, for example, consents, alteration timestamp, disk space amount, namespace portion, and times access. NameNode maps the total structure of the file system into memory. Two image files and changes are utilized for tirelessness amid reboots.

- **The image file contains the nodes and the list of blocks that characterize the metadata.**
- **The alter file contains the progressions made to the contents of the picture file.**

At the point when NameNode is begun, the image file is stacked, and after that the contents of the change file is connected to recover the most recent state of the file system. The main issue

with this is, after some time, the altering file develops and consumes all the disk space, which causes a slowdown in the reboot procedure. In the event that the hadoop cluster has not been restarted for a considerable length of time, there will be a great deal of downtime as the measure of the altered file increments. That is when Secondary NameNode acts the hero. Secondary NameNode gets the picture and alters the main NameNode record at normal interims and burdens the image file and changes the logs in main memory by applying every operation of the adjusted log file to the picture. Secondary NameNode duplicates the new image file to the main NameNode node and furthermore refreshes the changed time of the image file in the time file to take after when the image file has been refreshed.

## DataNode

DataNode deals with the state of a HDFS node and collaborates with the blocks. A DataNode can perform resource-intensive tasks, for example, semantic and linguistic analysis, statistics, and machine-learning tasks, and in addition intensive I/O tasks, for example, clustering, data import, and that's only the tip of the iceberg data export, search, decompression and ordering. A DataNode needs a ton of I/O for data processing and transfer.

Toward the starting, each DataNode interfaces with the NameNode and consults to confirm the namespace ID and programming adaptation of the DataNode. In the event that one of them doesn't coordinate, the DataNode stops naturally. A DataNode checks for block replicas in its property by sending a block answer to the NameNode. When the DataNode is spared, the principal block report is sent. DataNode sends a pulse to the NameNode like clockwork to affirm that DataNode is working and block replicas are accessible.

## Role of Distributed Computation:

The core of the Hadoop dispersed computing stage is its Java Hadoop MapReduce-based programming paradigm. Map or Reduce is a specific sort of focused non-cyclic chart that can be connected to an extensive variety of business utilize cases. The map work changes the data into key-esteem sets, and the keys are arranged when a rollback work is connected to combine the key-based esteems into a solitary output.

The execution of a MapReduce job starts when the client sends the job arrangement to the job tracker that indicates the map, joins and decreases the functions and the area of the input and output data. After getting the job arrangement, the job tracker distinguishes the quantity of divisions in light of the input way and chooses Task Tracker in view of the vicinity of its system to the data sources. Job Tracker sends a request to the chose Task Trackers.

The stage processing of the map starts when the task tracker removes the input data from the divisions. The map work is conjured for each record parsed by "Input Format" which produces enter esteem matches in the memory support. The memory cradle is grouped into various lessening nodes by summoning the joining capacity. When you play out the map task, the job tracker informs the job tracker. At the point when all task follows are finished, Job Tracker advises the chose task followers to start the reduction stage. Task Tracker peruses the area files and sorts the key-esteem sets for each key. At that point, the reduction work that gathers amassed values into the output file.

### **III. Method:**

#### **Procedure:**

1. Sudo su  
sudo is intended to run a solitary charge with root benefits. Be that as it may, not at all like su it prompts you for the watchword of the present client. This client must be in the sudoers document (or a gathering that is in the sudoers record).
2. ssh localhost  
Setting up a protected association. The charge implies that an association is directed to the possess machine, to the present client.
3. hadoop namenode -format  
Formatting the name node to start a new environment. NameNode is the centerpiece of HDFS. NameNode is also known as the Master. NameNode only stores the metadata of HDFS – the directory tree of all files in the file system, and tracks the files across the cluster. NameNode does not store the actual data or the dataset. The data itself is actually stored in the DataNodes.
4. start-all.sh  
Start all the nodes. The \$HADOOP\_INSTALL/hadoop/bin directory contains some scripts used to launch Hadoop DFS and Hadoop Map/Reduce daemons. These are:  
start-dfs.sh - Starts the Hadoop DFS daemons, the namenode, datanodes, the jobtracker and tasktrackers.

5. `jps`

Shows all the working nodes. The `jps` tool lists the instrumented HotSpot Java Virtual Machines (JVMs) on the target system. The tool is limited to reporting information on JVMs for which it has the access permissions.

If `jps` is run without specifying a `15adoop`, it will look for instrumented JVMs on the local host. If started with a `15adoop`, it will look for JVMs on the indicated host, using the specified protocol and port. A `jstatd` process is assumed to be running on the target host.

The list of JVMs produced by the `jps` command may be limited by the permissions granted to the principal running the command. The command will only list the JVMs for which the principle has access rights as determined by operating system specific access control mechanisms.

6. `15adoop dfsadmin -report`

Describes the health and other required aspects.

7. `15adoop dfs -lsr /`

Shows all the files in the hdfs. HDFS supports highly-available (HA) namenode services and wire compatibility. These two capabilities make it feasible to upgrade HDFS without incurring HDFS downtime. In order to upgrade a HDFS cluster without downtime, the cluster must be setup with HA.

If there is any new feature which is enabled in new software release, may not work with old software release after upgrade. In such cases upgrade should be done by following steps.

- Disable new feature.
- Upgrade the cluster.
- Enable the new feature.

8. `15adoop dfs -mkdir test`

Makes a new folder. In computer hardware and software development, testing is used at key checkpoints in the overall process to determine whether objectives are being met. For example, in software development, product objectives are sometimes tested by product user representatives. The `mkdir` command is used to create new directories.

A directory, referred to as a folder in some operating systems, appears to the user as a container for other directories and files.

9. `16adoop dfs -put abc.txt /user/test`  
Adds the test file to generate the training data. Copy single src, or multiple srcs from local file system to the destination file system. Also reads input from stdin and writes to destination file system.
10. `16adoop jar NB_web.jar Template /user/test/abc.txt /user/test/out1`
11.  
Generates training data. Runs a jar file. Users can bundle their Map Reduce code in a jar file and execute it using this command.
12. `16adoop dfs -cat /user/test/out1/part-r-00000`  
Shows the output file.
13. `16adoop dfs -rmr /`  
Formats the hdfs. Recursive version of delete. If the `-skipTrash` option is specified, the trash, if enabled, will be bypassed and the specified file(s) deleted immediately. This can be useful when it is necessary to delete files from an over-quota directory.
14. `stop-all.sh`  
Stops all the nodes. Used to stop 16adoop daemons all at once. Issuing it on the master machine will start/stop the daemons on all the nodes of a cluster. Deprecated as you have already noticed.
15. Exit  
exit is a command used in many operating system command line shells and scripting languages. The command causes the shell or program to terminate. If performed within an interactive command shell, the user is logged out of their current session, and/or user's current console or terminal connection is disconnected.

## **IV. Code:**

### ***JAR FILE:***

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



```

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
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 Navie {

    public static void main(String args[]) throws Exception {

        Configuration conf = new Configuration();

        Job job = Job.getInstance(conf, "Navie");

        job.setJarByClass(Navie.class);

        job.setMapperClass(TmpltMapper.class);
        job.setCombinerClass(TmpltCombiner.class);
        job.setReducerClass(TmpltReducer.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);
    }

    public static class TmpltMapper extends
        Mapper<LongWritable, Text, Text, IntWritable> {
        @Override
        public void map(LongWritable key, Text value, Mapper.Context
context)
            throws IOException, InterruptedException {
            context.write(new Text("MKEY"), value);
        }
    }

    public static class TmpltCombiner extends
        Reducer<Text, Text, Text, Text> {
        @Override
        public void reduce(Text key, Iterable<Text> values,
            Context context) throws IOException, InterruptedException {
            Iterator<Text> itr = values.iterator();
            Vector<String> age = new Vector<String>();
            Vector<String> inc = new Vector<String>();
            Vector<String> stu = new Vector<String>();
            Vector<String> crd = new Vector<String>();

```

```

Vector<String> buy = new Vector<String>();
while (itr.hasNext()) {
    final String text = itr.next().toString();
    StringTokenizer t = new StringTokenizer(text, ",");

    age.add(t.nextToken());
    inc.add(t.nextToken());
    stu.add(t.nextToken());
    crd.add(t.nextToken());
    buy.add(t.nextToken());
}
int tl=buy.size();
int yc=0;
int nc=0;
int a1yc=0;
int a1nc=0;
int a2yc=0;
int a2nc=0;
int a3yc=0;
int a3nc=0;
int hyc=0;
int hnc=0;
int myc=0;
int mnc=0;
int lyc=0;
int lnc=0;
int yyc=0;
int ync=0;
int nyc=0;
int nnc=0;
int fyc=0;
int fnc=0;
int eyc=0;
int enc=0;
for (int i=0;i<tl; i++){
    if(buy.get(i).equals("ys"))    yc++;
    if(buy.get(i).equals("no"))    nc++;

    if((age.get(i).equals("<=30"))&(buy.get(i).equals("ys"))    a1yc++;
    if((age.get(i).equals("<=30"))&(buy.get(i).equals("no"))    a1nc++;
    if((age.get(i).equals("31..40"))&(buy.get(i).equals("ys"))    a2yc++;
    if((age.get(i).equals("31..40"))&(buy.get(i).equals("no"))    a2nc++;
    if((age.get(i).equals(">40"))&(buy.get(i).equals("ys"))    a3yc++;
    if((age.get(i).equals(">40"))&(buy.get(i).equals("no"))    a3nc++;
    if((inc.get(i).equals("hi"))&(buy.get(i).equals("ys"))    hyc++;
    if((inc.get(i).equals("hi"))&(buy.get(i).equals("no"))    hnc++;
    if((inc.get(i).equals("md"))&(buy.get(i).equals("ys"))    myc++;

    if((inc.get(i).equals("md"))&(buy.get(i).equals("no"))    mnc++;
    if((inc.get(i).equals("lw"))&(buy.get(i).equals("ys"))    lyc++;
    if((inc.get(i).equals("lw"))&(buy.get(i).equals("no"))    lnc++;
    if((stu.get(i).equals("ys"))&(buy.get(i).equals("ys"))    yyc++;
    if((stu.get(i).equals("ys"))&(buy.get(i).equals("no"))    ync++;
    if((stu.get(i).equals("no"))&(buy.get(i).equals("ys"))    nyc++;
    if((stu.get(i).equals("no"))&(buy.get(i).equals("no"))    nnc++;
    if((crd.get(i).equals("fr"))&(buy.get(i).equals("ys"))    fyc++;

```

```

        if((crd.get(i).equals("fr")) & (buy.get(i).equals("no")))    fnc++;
        if((crd.get(i).equals("ex")) & (buy.get(i).equals("ys")))    eyc++;
        if((crd.get(i).equals("ex")) & (buy.get(i).equals("no")))    enc++;
    }

    context.write(key, new
Text(yc+"_"+nc+"_"+a1yc+"_"+a1nc+"_"+a2yc+"_"+a2nc+"_"+a3yc+"_"+a3nc+"_"+hyc+
"_"+hnc+"_"+myc+"_"+mnc+"_"+lyc+"_"+lnc+"_"+yyc+"_"+ync+"_"+nyc+"_"+nnc+"_"+f
yc+"_"+fnc+"_"+eyc+"_"+enc+"_"+tl));
}
}

```

```

    public static class TmplTReducer extends
        Reducer<Text, Text, Text, Text> {
        @Override
        public void reduce(Text key, Iterable<Text> values,
            Context context) throws IOException,
InterruptedException {
            Iterator<Text> itr = values.iterator();
            int yc=0;
            int nc=0;
            int a1yc=0;
            int a1nc=0;
            int a2yc=0;
            int a2nc=0;
            int a3yc=0;
            int a3nc=0;
            int hyc=0;
            int hnc=0;
            int myc=0;
            int mnc=0;
            int lyc=0;
            int lnc=0;
            int yyc=0;
            int ync=0;
            int nyc=0;
            int nnc=0;
            int fyc=0;
            int fnc=0;
            int eyc=0;
            int enc=0;
            int tl=0;

            while (itr.hasNext()) {
                final String text = itr.next().toString();
                StringTokenizer st = new StringTokenizer(text, "_");
                yc+=Integer.parseInt(st.nextToken());
                nc+=Integer.parseInt(st.nextToken());
                a1yc+=Integer.parseInt(st.nextToken());
                a1nc+=Integer.parseInt(st.nextToken());
                a2yc+=Integer.parseInt(st.nextToken());
                a2nc+=Integer.parseInt(st.nextToken());
                a3yc+=Integer.parseInt(st.nextToken());
                a3nc+=Integer.parseInt(st.nextToken());
                hyc+=Integer.parseInt(st.nextToken());
            }
        }
    }
}

```

```

        hnc+=Integer.parseInt(st.nextToken());
        myc+=Integer.parseInt(st.nextToken());
        mnc+=Integer.parseInt(st.nextToken());
        lyc+=Integer.parseInt(st.nextToken());
        lnc+=Integer.parseInt(st.nextToken());
        yyc+=Integer.parseInt(st.nextToken());
        ync+=Integer.parseInt(st.nextToken());
        nyc+=Integer.parseInt(st.nextToken());
        nnc+=Integer.parseInt(st.nextToken());
        fyc+=Integer.parseInt(st.nextToken());
        fnc+=Integer.parseInt(st.nextToken());
        eyc+=Integer.parseInt(st.nextToken());
        enc+=Integer.parseInt(st.nextToken());
        tl += Integer.parseInt(st.nextToken());

    }
    context.write(key, new Text(yc+" "+nc+" "+alyc+" "+alnc+"
+a2yc+" "+a2nc+" "+a3yc+" "+a3nc+" "+hyc+" "+hnc+" "+myc+" "+mnc+" "+lyc+"
"+lnc+" "+yyc+" "+ync+" "+nyc+" "+nnc+" "+fyc+" "+fnc+" "+eyc+" "+enc+"
"+tl));

    }

}

```

## ***DESKTOP.JAVA***

```

import java.util.StringTokenizer;

public class Desktop {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        String line = "75 51 25 20 21 15 29 16 22 22 28 16 25 13 25 31 50
20 51 38 24 13 126";
        StringTokenizer st = new StringTokenizer(line, " ");

        int yc=Integer.parseInt(st.nextToken());
        int nc=Integer.parseInt(st.nextToken());
        int alyc=Integer.parseInt(st.nextToken());
        int alnc=Integer.parseInt(st.nextToken());
        int a2yc=Integer.parseInt(st.nextToken());
        int a2nc=Integer.parseInt(st.nextToken());
        int a3yc=Integer.parseInt(st.nextToken());
        int a3nc=Integer.parseInt(st.nextToken());
        int hyc=Integer.parseInt(st.nextToken());
        int hnc=Integer.parseInt(st.nextToken());
        int myc=Integer.parseInt(st.nextToken());
        int mnc=Integer.parseInt(st.nextToken());
        int lyc=Integer.parseInt(st.nextToken());
        int lnc=Integer.parseInt(st.nextToken());
        int yyc=Integer.parseInt(st.nextToken());
        int ync=Integer.parseInt(st.nextToken());
    }
}

```

```

int nyc=Integer.parseInt(st.nextToken());
int nnc=Integer.parseInt(st.nextToken());
int fyc=Integer.parseInt(st.nextToken());
int fnc=Integer.parseInt(st.nextToken());
int eyc=Integer.parseInt(st.nextToken());
int enc=Integer.parseInt(st.nextToken());
int tl = Integer.parseInt(st.nextToken());

String[] test = new String[]{">40","lw","no","fr"};
double prys= (double)yc/tl; //P(yes)
double prno= (double)nc/tl; //P(no)
for (int i = 0; i < test.length; i++) {
String vl = test[i];
if(vl.equals("<=30")){
prys *= (double)a1yc/yc;//P(<=30/yes)
prno *= (double)a1nc/nc; //P(<=30/no)
}else if(vl.equals("31..40")){
prys *= (double)a2yc/yc;//P(31..40/yes)
prno *= (double)a2nc/nc; //P(31..40/no)
}else if(vl.equals(">40")){
prys *= (double)a3yc/yc;//P(>40/yes)
prno *= (double)a3nc/nc; //P(>40/no)
}else if(vl.equals("hi")){
prys *= (double)hyc/yc;//P(high/yes)
prno *= (double)hnc/nc; //P(high/no)
}else if(vl.equals("md")){
prys *= (double)myc/yc;//P(medium/yes)
prno *= (double)mnc/nc; //P(medium/no)
}else if(vl.equals("lw")){
prys *= (double)lyc/yc;//P(low/yes)
prno *= (double)lnc/nc; //P(low/no)

}else if(vl.equals("ys")){
prys *= (double)yyc/yc;//P(yes/yes)
prno *= (double)ync/nc; //P(yes/no)
}else if(vl.equals("no")){
prys *= (double)nyc/yc;//P(no/yes)
prno *= (double)nnc/nc; //P(no/no)
}else if(vl.equals("fr")){
prys *= (double)fyc/yc;//P(fair/yes)
prno *= (double)fnc/nc; //P(fair/no)
}else if(vl.equals("ex")){
prys *= (double)eyc/yc;//P(excellent/yes)
prno *= (double)enc/nc; //P(excellent/no)
}
}

String result = prys>prno? "will buy\n": "will not buy \n";
System.out.println(result+"yes:"+prys+" no:"+prno);

if(prys>prno)
    System.out.println("The customer will buy the
product");
else
    System.out.println("The customer will not buy the
product");
}
}

```

## V. Results:

The jar file is executed and the model is generated with which we can predict the probability of each attribute with the decision whether the product is bought or not is decided.

The desktop model is used as a test case file to predict the probability and come up with the result whether the customer buys the product or not along with the discount percentage to be offered to the customer.

```
vishal@hena-Inspiron-3521:~$ hadoop dfs -put test_nb.txt /user/vishal/test
vishal@hena-Inspiron-3521:~$ hadoop dfs -lsr /
drwxr-xr-x  - vishal supergroup      0 2017-10-11 22:34 /app
drwxr-xr-x  - vishal supergroup      0 2017-10-11 22:34 /app/hadoop
drwxr-xr-x  - vishal supergroup      0 2017-10-11 22:34 /app/hadoop/tmp
drwxr-xr-x  - vishal supergroup      0 2017-10-11 22:34 /app/hadoop/tmp/mapred
drwx----- - vishal supergroup      0 2017-10-11 22:34 /app/hadoop/tmp/mapred/system
-rw-----  1 vishal supergroup      4 2017-10-11 22:34 /app/hadoop/tmp/mapred/system/jobtracker.info
-rw-r--r--  1 vishal supergroup    241 2017-10-11 22:37 /test
drwxr-xr-x  - vishal supergroup      0 2017-10-11 22:35 /user
drwxr-xr-x  - vishal supergroup      0 2017-10-11 22:35 /user/vishal
drwxr-xr-x  - vishal supergroup      0 2017-10-11 22:37 /user/vishal/test
-rw-r--r--  1 vishal supergroup    241 2017-10-11 22:37 /user/vishal/test/test_nb.txt
vishal@hena-Inspiron-3521:~$
```

```
vishal@hena-Inspiron-3521:~$ hadoop jar NB_web1.jar Template /user/vishal/test/test_nb.txt /user/vishal/out1
17/10/11 22:38:58 WARN mapred.JobClient: Use GenericOptionsParser for parsing the arguments. Applications should implement Tool for the same.
17/10/11 22:38:58 INFO input.FileInputFormat: Total input paths to process : 1
17/10/11 22:38:58 INFO util.NativeCodeLoader: Loaded the native-hadoop library
17/10/11 22:38:58 WARN snappy.LoadSnappy: Snappy native library not loaded
17/10/11 22:38:59 INFO mapred.JobClient: Running job: job_201710112234_0001
17/10/11 22:39:00 INFO mapred.JobClient:  map 0% reduce 0%
17/10/11 22:39:06 INFO mapred.JobClient:  map 100% reduce 0%
17/10/11 22:39:15 INFO mapred.JobClient:  map 100% reduce 33%
17/10/11 22:39:16 INFO mapred.JobClient:  map 100% reduce 100%
17/10/11 22:39:17 INFO mapred.JobClient: Job complete: job_201710112234_0001
17/10/11 22:39:17 INFO mapred.JobClient: Counters: 29
17/10/11 22:39:17 INFO mapred.JobClient:   Job Counters
17/10/11 22:39:17 INFO mapred.JobClient:     Launched reduce tasks=1
17/10/11 22:39:17 INFO mapred.JobClient:     SLOTS_MILLIS_MAPS=6349
17/10/11 22:39:17 INFO mapred.JobClient:     Total time spent by all reduces waiting after reserving slots (ms)=0
17/10/11 22:39:17 INFO mapred.JobClient:     Total time spent by all maps waiting after reserving slots (ms)=0
17/10/11 22:39:17 INFO mapred.JobClient:     Launched map tasks=1
17/10/11 22:39:17 INFO mapred.JobClient:     Data-local map tasks=1
17/10/11 22:39:17 INFO mapred.JobClient:     SLOTS_MILLIS_REDUCE=9385
17/10/11 22:39:17 INFO mapred.JobClient:   File Output Format Counters
17/10/11 22:39:17 INFO mapred.JobClient:     Bytes Written=52
17/10/11 22:39:17 INFO mapred.JobClient:   FileSystemCounters
17/10/11 22:39:17 INFO mapred.JobClient:     FILE_BYTES_READ=60
17/10/11 22:39:17 INFO mapred.JobClient:     HDFS_BYTES_READ=357
17/10/11 22:39:17 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=109386
17/10/11 22:39:17 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=52
17/10/11 22:39:17 INFO mapred.JobClient:   File Input Format Counters
17/10/11 22:39:17 INFO mapred.JobClient:     Bytes Read=241
17/10/11 22:39:17 INFO mapred.JobClient:   Map-Reduce Framework
17/10/11 22:39:17 INFO mapred.JobClient:     Map output materialized bytes=60
17/10/11 22:39:17 INFO mapred.JobClient:     Map input records=14
17/10/11 22:39:17 INFO mapred.JobClient:     Reduce shuffle bytes=60
17/10/11 22:39:17 INFO mapred.JobClient:     Spilled Records=2
```



```

17/10/11 22:39:17 INFO mapred.JobClient: Launched map tasks=1
17/10/11 22:39:17 INFO mapred.JobClient: Data-local map tasks=1
17/10/11 22:39:17 INFO mapred.JobClient: SLOTS_MILLIS_REDUCES=9385
17/10/11 22:39:17 INFO mapred.JobClient: File Output Format Counters
17/10/11 22:39:17 INFO mapred.JobClient: Bytes Written=52
17/10/11 22:39:17 INFO mapred.JobClient: FileSystemCounters
17/10/11 22:39:17 INFO mapred.JobClient: FILE_BYTES_READ=60
17/10/11 22:39:17 INFO mapred.JobClient: HDFS_BYTES_READ=357
17/10/11 22:39:17 INFO mapred.JobClient: FILE_BYTES_WRITTEN=109386
17/10/11 22:39:17 INFO mapred.JobClient: HDFS_BYTES_WRITTEN=52
17/10/11 22:39:17 INFO mapred.JobClient: File Input Format Counters
17/10/11 22:39:17 INFO mapred.JobClient: Bytes Read=241
17/10/11 22:39:17 INFO mapred.JobClient: Map-Reduce Framework
17/10/11 22:39:17 INFO mapred.JobClient: Map output materialized bytes=60
17/10/11 22:39:17 INFO mapred.JobClient: Map input records=14
17/10/11 22:39:17 INFO mapred.JobClient: Reduce shuffle bytes=60
17/10/11 22:39:17 INFO mapred.JobClient: Spilled Records=2
17/10/11 22:39:17 INFO mapred.JobClient: Map output bytes=311
17/10/11 22:39:17 INFO mapred.JobClient: Total committed heap usage (bytes)=218103808
17/10/11 22:39:17 INFO mapred.JobClient: CPU time spent (ms)=1820
17/10/11 22:39:17 INFO mapred.JobClient: Combine input records=14
17/10/11 22:39:17 INFO mapred.JobClient: SPLIT_RAW_BYTES=116
17/10/11 22:39:17 INFO mapred.JobClient: Reduce input records=1
17/10/11 22:39:17 INFO mapred.JobClient: Reduce input groups=1
17/10/11 22:39:17 INFO mapred.JobClient: Combine output records=1
17/10/11 22:39:17 INFO mapred.JobClient: Physical memory (bytes) snapshot=279412736
17/10/11 22:39:17 INFO mapred.JobClient: Reduce output records=1
17/10/11 22:39:17 INFO mapred.JobClient: Virtual memory (bytes) snapshot=1602297856
17/10/11 22:39:17 INFO mapred.JobClient: Map output records=14
vishal@hema-Inspiron-3521:~$

```

```

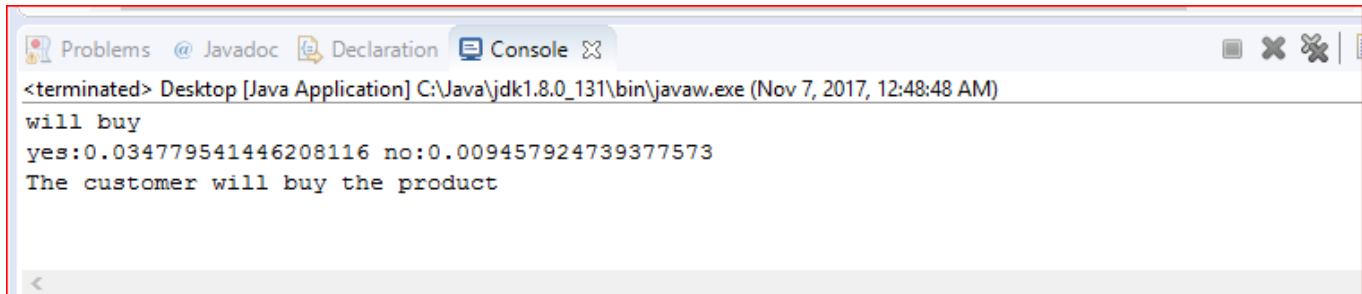
vishal@hema-Inspiron-3521:~$ hadoop dfs -lsr /
drwxr-xr-x - vishal supergroup 0 2017-10-11 22:34 /app
drwxr-xr-x - vishal supergroup 0 2017-10-11 22:34 /app/hadoop
drwxr-xr-x - vishal supergroup 0 2017-10-11 22:34 /app/hadoop/tmp
drwxr-xr-x - vishal supergroup 0 2017-10-11 22:38 /app/hadoop/tmp/napred
drwxr-xr-x - vishal supergroup 0 2017-10-11 22:38 /app/hadoop/tmp/napred/staging
drwxr-xr-x - vishal supergroup 0 2017-10-11 22:38 /app/hadoop/tmp/napred/staging/vishal
drwx----- - vishal supergroup 0 2017-10-11 22:39 /app/hadoop/tmp/napred/staging/vishal/.staging
drwx----- - vishal supergroup 0 2017-10-11 22:39 /app/hadoop/tmp/napred/system
-rw----- 1 vishal supergroup 4 2017-10-11 22:34 /app/hadoop/tmp/napred/system/jobtracker.info
-rw-r--r-- 1 vishal supergroup 241 2017-10-11 22:37 /test
drwxr-xr-x - vishal supergroup 0 2017-10-11 22:35 /user
drwxr-xr-x - vishal supergroup 0 2017-10-11 22:38 /user/vishal
drwxr-xr-x - vishal supergroup 0 2017-10-11 22:39 /user/vishal/out1
-rw-r--r-- 1 vishal supergroup 0 2017-10-11 22:39 /user/vishal/out1/_SUCCESS
drwxr-xr-x - vishal supergroup 0 2017-10-11 22:38 /user/vishal/out1/_logs
drwxr-xr-x - vishal supergroup 0 2017-10-11 22:38 /user/vishal/out1/_logs/history
-rw-r--r-- 1 vishal supergroup 13717 2017-10-11 22:38 /user/vishal/out1/_logs/history/job_201710112234_0001_1507741739157_vishal_Template
-rw-r--r-- 1 vishal supergroup 47055 2017-10-11 22:38 /user/vishal/out1/_logs/history/job_201710112234_0001_conf.xml
-rw-r--r-- 1 vishal supergroup 52 2017-10-11 22:39 /user/vishal/out1/part-r-00000
drwxr-xr-x - vishal supergroup 0 2017-10-11 22:37 /user/vishal/test
-rw-r--r-- 1 vishal supergroup 241 2017-10-11 22:37 /user/vishal/test/test_nb.txt

```

```

vishal@hema-Inspiron-3521:~$ hadoop dfs -cat /user/vishal/out1/part-r-00000
MKEY 9 5 2 3 4 0 3 2 2 2 4 2 3 1 6 1 3 4 6 2 3 3 14
vishal@hema-Inspiron-3521:~$

```

A screenshot of a Java IDE's console window. The title bar shows tabs for 'Problems', '@ Javadoc', 'Declaration', and 'Console'. The console text reads: '<terminated> Desktop [Java Application] C:\Java\jdk1.8.0\_131\bin\javaw.exe (Nov 7, 2017, 12:48:48 AM)', 'will buy', 'yes:0.034779541446208116 no:0.009457924739377573', and 'The customer will buy the product'.

```
<terminated> Desktop [Java Application] C:\Java\jdk1.8.0_131\bin\javaw.exe (Nov 7, 2017, 12:48:48 AM)
will buy
yes:0.034779541446208116 no:0.009457924739377573
The customer will buy the product
```

## **VI. References:**

- [1] Agarwal R. and Srikant R., "Fast algorithms for mining association rules", VLDB'94, Chile, pp. 487–499, 1994. `
- [2] Anderson C., Domingos P., Weld D. S., "Relational Markov Models and their Application to Adaptive Web Navigation", Proceedings of the 8th ACM SIGKDD Conference, Canada, August 2002.
- [3] Bamshad Mobasher, Robert Cooley, Jaideep Srivastava," Creating Adaptive Web Sites Through Usage-Based Clustering of URLs", proceedings of the 1999 workshop on knowledge and data engineering, pp 19, 1999.
- [4] Bruha I., "From machine learning to knowledge discovery: Survey of preprocessing and postprocessing" Intelligent Data Analysis, Vol. 4, pp. 363-374, 2000.
- [5] Buchner A. and Mulvenna M. D., "Discovering Internet Marketing Intelligence through Online Analytical Web Usage Mining", Proceedings of the ACM SIGMOD, Intl.Conf. on Management of Data (SIGMOD'99), pp. 54– 61, 1999.
- [6] Burges C., "A tutorial on support vector machines for pattern recognition", Data Mining and Knowledge Discovery, Vol. 2, pp. 1-47, 1998.
- [7] Cheng D., Kannan R., Vempala S. and Wang G., "A divide-and-merge methodology for clustering", ACM SIGMOD, pp. 196–212, 2005.
- [8] Cooley R. , Mobasher B., and Srivastava J. , "Data Preparation for Mining World Wide Web Browsing Patterns", Knowledge and Information Systems, vol. 1(1), pp. 5–32, 1999.
- [9] Cristianini N. and Shawe-Taylor J., "An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods", Cambridge University Press, Cambridge, 2000.
- [10] Deshpande M., Karypis G., "Selective Markov Models for Predicting Web-Page Accesses", Proceedings of the 1st SIAM International Conference on Data Mining, 2004.



- [11] Dhruv Gupta, Mark Digiovanni, Hiro Narita, and Ken Goldberg, "Jester 2.0 : Evaluation of a New Linear Time Collaborative Filtering Algorithm", SIGIR '99, Berkeley, CA, USA, ACM, 1999.
- [12] Domingos P. and Pazzani M. , "On the optimality of the simple Bayesian classifier under zero-one loss". Machine Learning, pp.103-130, 1997.
- [13] [https://hadoop.apache.org/docs/r1.2.1/mapred\\_tutorial.pdf](https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.pdf)
- [14] [https://hadoop.apache.org/docs/r1.2.1/hdfs\\_design.pdf](https://hadoop.apache.org/docs/r1.2.1/hdfs_design.pdf)
- [15] IEEE Journal: "Big data management processing with Hadoop MapReduce and spark technology: A comparison".