

5

Learning Data Analytics with R and Hadoop

In the previous chapters we learned about the installation, configuration, and integration of R and Hadoop.

In this chapter, we will learn how to perform data analytics operations over an integrated R and Hadoop environment. Since this chapter is designed for data analytics, we will understand this with an effective data analytics cycle.

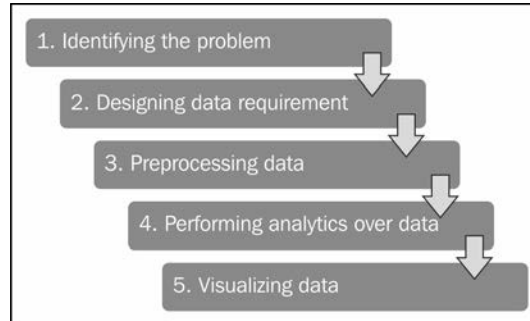
In this chapter we will learn about:

- Understanding the data analytics project life cycle
- Understanding data analytics problems

Understanding the data analytics project life cycle

While dealing with the data analytics projects, there are some fixed tasks that should be followed to get the expected output. So here we are going to build a data analytics project cycle, which will be a set of standard data-driven processes to lead data to insights effectively. The defined data analytics processes of a project life cycle should be followed by sequences for effectively achieving the goal using input datasets. This data analytics process may include identifying the data analytics problems, designing, and collecting datasets, data analytics, and data visualization.

The data analytics project life cycle stages are seen in the following diagram:



Let's get some perspective on these stages for performing data analytics.

Identifying the problem

Today, business analytics trends change by performing data analytics over web datasets for growing business. Since their data size is increasing gradually day by day, their analytical application needs to be scalable for collecting insights from their datasets.

With the help of web analytics, we can solve the business analytics problems. Let's assume that we have a large e-commerce website, and we want to know how to increase the business. We can identify the important pages of our website by categorizing them as per popularity into high, medium, and low. Based on these popular pages, their types, their traffic sources, and their content, we will be able to decide the roadmap to improve business by improving web traffic, as well as content.

Designing data requirement

To perform the data analytics for a specific problem, it needs datasets from related domains. Based on the domain and problem specification, the data source can be decided and based on the problem definition; the data attributes of these datasets can be defined.

For example, if we are going to perform social media analytics (problem specification), we use the data source as Facebook or Twitter. For identifying the user characteristics, we need user profile information, likes, and posts as data attributes.

Preprocessing data

In data analytics, we do not use the same data sources, data attributes, data tools, and algorithms all the time as all of them will not use data in the same format. This leads to the performance of data operations, such as data cleansing, data aggregation, data augmentation, data sorting, and data formatting, to provide the data in a supported format to all the data tools as well as algorithms that will be used in the data analytics.

In simple terms, preprocessing is used to perform data operation to translate data into a fixed data format before providing data to algorithms or tools. The data analytics process will then be initiated with this formatted data as the input.

In case of Big Data, the datasets need to be formatted and uploaded to **Hadoop Distributed File System (HDFS)** and used further by various nodes with Mappers and Reducers in Hadoop clusters.

Performing analytics over data

After data is available in the required format for data analytics algorithms, data analytics operations will be performed. The data analytics operations are performed for discovering meaningful information from data to take better decisions towards business with data mining concepts. It may either use descriptive or predictive analytics for business intelligence.

Analytics can be performed with various machine learning as well as custom algorithmic concepts, such as regression, classification, clustering, and model-based recommendation. For Big Data, the same algorithms can be translated to MapReduce algorithms for running them on Hadoop clusters by translating their data analytics logic to the MapReduce job which is to be run over Hadoop clusters. These models need to be further evaluated as well as improved by various evaluation stages of machine learning concepts. Improved or optimized algorithms can provide better insights.

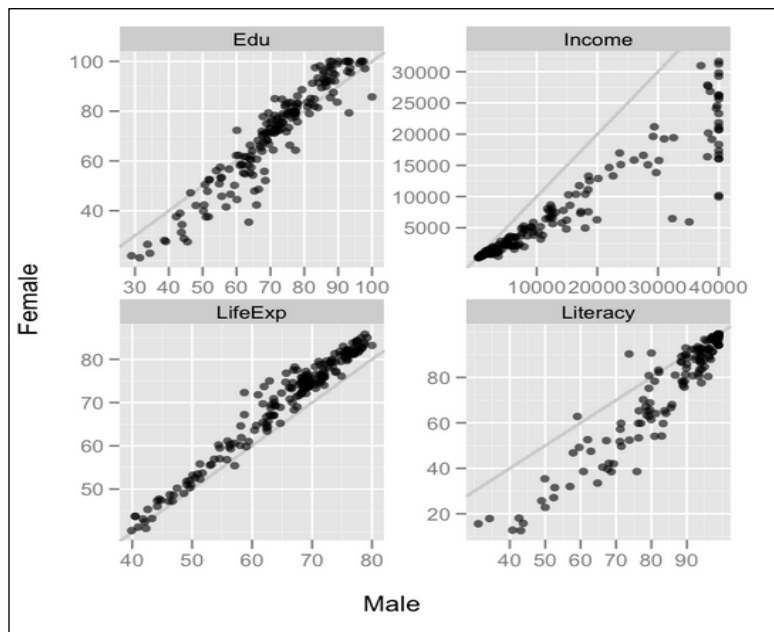
Visualizing data

Data visualization is used for displaying the output of data analytics. Visualization is an interactive way to represent the data insights. This can be done with various data visualization softwares as well as R packages. R has a variety of packages for the visualization of datasets. They are as follows:

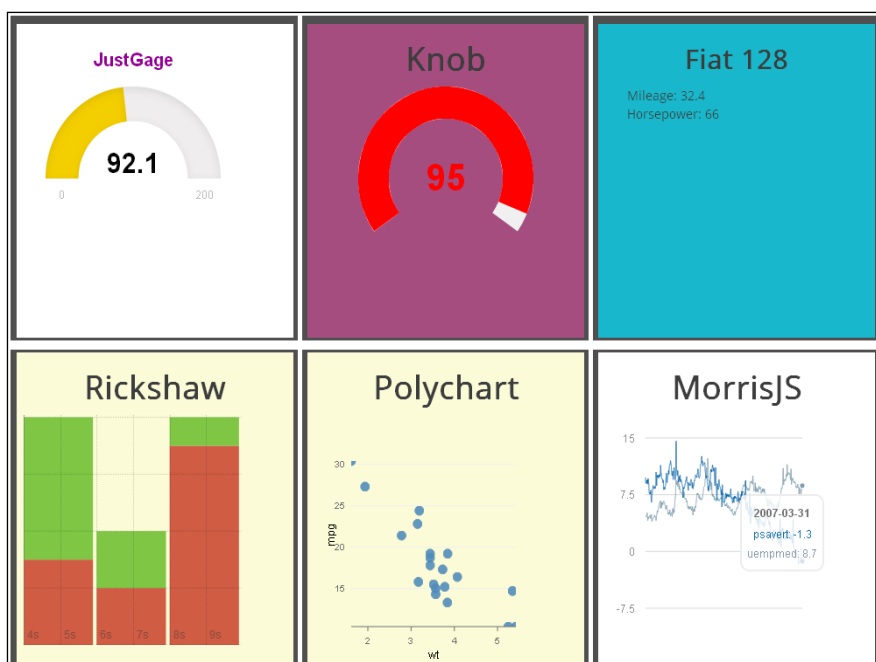
- **ggplot2:** This is an implementation of the Grammar of Graphics by *Dr. Hadley Wickham* (<http://had.co.nz/>). For more information refer <http://cran.r-project.org/web/packages/ggplot2/>.
- **rCharts:** This is an R package to create, customize, and publish interactive JavaScript visualizations from R by using a familiar lattice-style plotting interface by *Markus Gesmann* and *Diego de Castillo*. For more information refer <http://ramnathv.github.io/rCharts/>.

Some popular examples of visualization with R are as follows:

- **Plots for facet scales (ggplot):** The following figure shows the comparison of males and females with different measures; namely, education, income, life expectancy, and literacy, using ggplot:



- **Dashboard charts:** This is an `rCharts` type. Using this we can build interactive animated dashboards with R.



Understanding data analytics problems

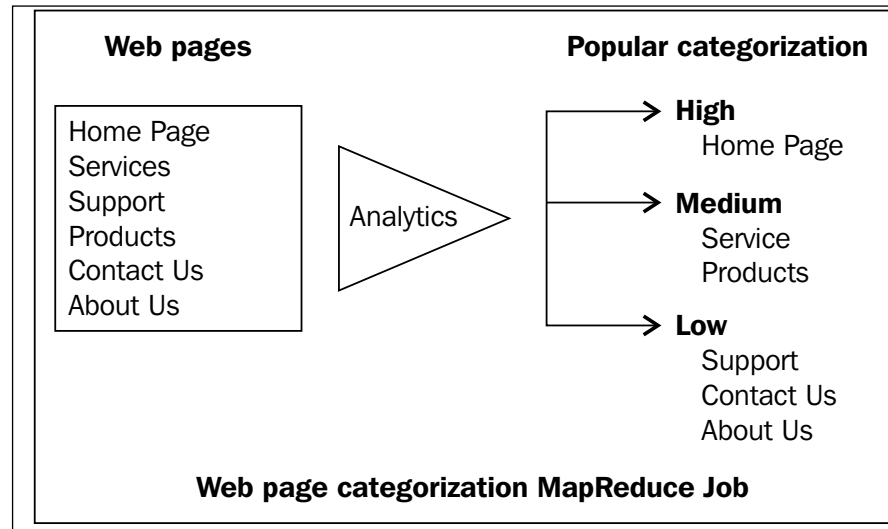
In this section, we have included three practical data analytics problems with various stages of data-driven activity with R and Hadoop technologies. These data analytics problem definitions are designed such that readers can understand how Big Data analytics can be done with the analytical power of functions, packages of R, and the computational powers of Hadoop.

The data analytics problem definitions are as follows:

- Exploring the categorization of web pages
- Computing the frequency of changes in the stock market
- Predicting the sale price of a blue book for bulldozers (case study)

Exploring web pages categorization

This data analytics problem is designed to identify the category of a web page of a website, which may categorized popularity wise as high, medium, or low (regular), based on the visit count of the pages. While designing the data requirement stage of the data analytics life cycle, we will see how to collect these types of data from Google Analytics.



Identifying the problem

As this is a web analytics problem, the goal of the problem is to identify the importance of web pages designed for websites. Based on this information, the content, design, or visits of the lower popular pages can be improved or increased.

Designing data requirement

In this section, we will be working with data requirement as well as data collection for this data analytics problem. First let's see how the requirement for data can be achieved for this problem.

Since this is a web analytics problem, we will use Google Analytics data source. To retrieve this data from Google Analytics, we need to have an existent Google Analytics account with web traffic data stored on it. To increase the popularity, we will require the visits information of all of the web pages. Also, there are many other attributes available in Google Analytics with respect to dimensions and metrics.

Understanding the required Google Analytics data attributes

The header format of the dataset to be extracted from Google Analytics is as follows:

```
date, source, pageTitle, pagePath
```

- date: This is the date of the day when the web page was visited
- source: This is the referral to the web page
- pageTitle: This is the title of the web page
- pagePath: This is the URL of the web page

Collecting data

As we are going to extract the data from Google Analytics, we need to use `RGoogleAnalytics`, which is an R library for extracting Google Analytics datasets within R. To extract data, you need this plugin to be installed in R. Then you will be able to use its functions.

The following is the code for the extraction process from Google Analytics:

```
# Loading the RGoogleAnalytics library
require("RGoogleAnalytics")

# Step 1. Authorize your account and paste the access_token
query <- QueryBuilder()
access_token <- query$authorize()

# Step 2. Create a new Google Analytics API object
ga <- RGoogleAnalytics()

# To retrieve profiles from Google Analytics
ga.profiles <- ga$GetProfileData(access_token)

# List the GA profiles
ga.profiles

# Step 3. Setting up the input parameters
profile <- ga.profiles$id[3]
startdate <- "2010-01-08"
enddate <- "2013-08-23"
dimension <- "ga:date,ga:source,ga:pageTitle,ga:pagePath"
metric <- "ga:visits"
sort <- "ga:visits"
maxresults <- 100099
```

```
# Step 4. Build the query string, use the profile by setting its index
value
query$Init(start.date = startdate,
            end.date = enddate,
            dimensions = dimension,
            metrics = metric,

            max.results = maxresults,
            table.id = paste("ga:",profile,sep="",collapse=""),
            access_token=access_token)

# Step 5. Make a request to get the data from the API
ga.data <- ga$GetReportData(query)

# Look at the returned data
head(ga.data)
write.csv(ga.data,"webpages.csv", row.names=FALSE)
```

The preceding file will be available with the chapter contents for download.

Preprocessing data

Now, we have the raw data for Google Analytics available in a CSV file. We need to process this data before providing it to the MapReduce algorithm.

There are two main changes that need to be performed into the dataset:

- Query parameters needs to be removed from the column pagePath as follows:

```
pagePath <- as.character(data$pagePath)
pagePath <- strsplit(pagePath,"\\?")
pagePath <- do.call("rbind", pagePath)
pagePath <- pagePath [,1]
```

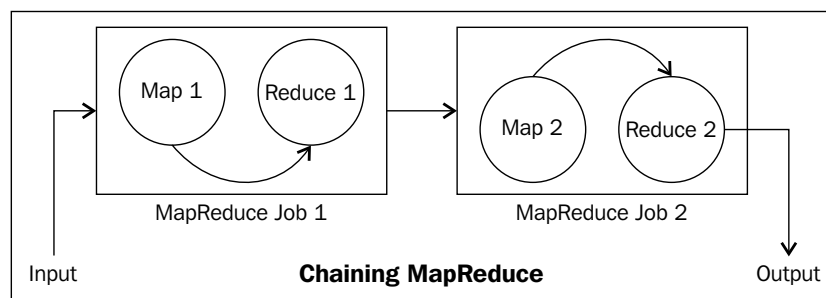
- The new CSV file needs to be created as follows:

```
data <- data.frame(source=data$source, pagePath=d,visits =)
write.csv(data, "webpages_mapreduce.csv" , row.names=FALSE)
```

Performing analytics over data

To perform the categorization over website pages, we will build and run the MapReduce algorithm with R and Hadoop integration. As already discussed in the *Chapter 2, Writing Hadoop MapReduce Programs*, sometimes we need to use multiple Mappers and Reducers for performing data analytics; this means using the chained MapReduce jobs.

In case of chaining MapReduce jobs, multiple Mappers and Reducers can communicate in such a way that the output of the first job will be assigned to the second job as input. The MapReduce execution sequence is described in the following diagram:



Chaining MapReduce

Now let's start with the programming task to perform analytics:

1. Initialize by setting Hadoop variables and loading the `rmr2` and `rhdfs` packages of the RHadoop libraries:

```
# setting up the Hadoop variables need by RHadoop
Sys.setenv(HADOOP_HOME="/usr/local/hadoop/")
Sys.setenv(HADOOP_CMD="/usr/local/hadoop/bin/hadoop")
```

```
# Loading the RHadoop libraries rmr2 and rhdfs
library(rmr2)
library(rhdfs)
```

```
# To initializing hdfs
hdfs.init()
```

2. Upload the datasets to HDFS:

```
# First uploading the data to R console,
webpages <- read.csv("/home/vigs/Downloads/webpages_mapreduce.
csv")
```

```
# saving R file object to HDFS,
webpages.hdfs <- to.dfs(webpages)
```

Now we will see the development of Hadoop MapReduce job 1 for these analytics. We will divide this job into Mapper and Reducer. Since, there are two MapReduce jobs, there will be two Mappers and Reducers. Also note that here we need to create only one file for both the jobs with all Mappers and Reducers. Mapper and Reducer will be established by defining their separate functions.

Let's see MapReduce job 1.

- **Mapper 1:** The code for this is as follows:

```
mapper1 <- function(k,v) {  
  
  # To storing pagePath column data in to key object  
  key <- v[2]  
  
  # To store visits column data into val object  
  Val <- v[3]  
  
  # emitting key and value for each row  
  keyval(key, val)  
}  
totalvisits <- sum(webpages$visits)
```

- **Reducer 1:** The code for this is as follows:

```
reducer1 <- function(k,v) {  
  
  # Calculating percentage visits for the specific URL  
  per <- (sum(v)/totalvisits)*100  
  # Identify the category of URL  
  if (per <33 )  
  {  
    val <- "low"  
  }  
  if (per >33 && per < 67)  
  {  
    val <- "medium"  
  }  
  if (per > 67)  
  {  
    val <- "high"  
  }  
  
  # emitting key and values  
  keyval(k, val)  
}
```

- **Output of MapReduce job 1:** The intermediate output for the information is shown in the following screenshot:

```

$key
                                pagePath
1                                /
2                                /abbranch.php
3                                /admission/
4                                /admission/
5                                /admission/diplomaadmissionpossibilities1.php
6                                /advertisewithus
7                                /bindingcollegedetails.php
8                                /bindingmpharmcollege.php
9                                /engineering-admission-possibilities
10                               /gtuadmissionhelpline-team
11                               /gujarat-degree-engineering-college
12                               /MBA-MCA/aes-institute-of-computer-studies
13 /medicalcollege/s-s-agarwal-college-of-nursing-navsari
14                               /meritcalc.php
15                               /merit-calculator
16                               /new_mba_college_list.php
17                               /newmecollege.php
18                               /newmpharmcollege.php
19                               /ourteam.php
20                               /search
21                               /search2.php
22                               /search3.php
23                               /search.php

$val
[1] "high" "low" "low" "low" "low" "low" "low" "low" "low" "low" "low" "low" "low" "low" "low" "low"
[17] "low" "low" "low" "low" "low" "low" "low"

```

The output in the preceding screenshot is only for information about the output of this MapReduce job 1. This can be considered an intermediate output where only 100 data rows have been considered from the whole dataset for providing output. In these rows, 23 URLs are unique; so the output has provided 23 URLs.

Let's see Hadoop MapReduce job 2:

- **Mapper 2:** The code for this is as follows:

```

#Mapper:
mapper2 <- function(k, v) {

# Reversing key and values and emitting them
  keyval(v,k)

}

```

- **Reducer 2:** The code for this is as follows:

```
key <- NA
val <- NULL
# Reducer:
reducer2 <- function(k, v) {

# for checking whether key-values are already assigned or not.
  if(is.na(key)) {
    key <- k
    val <- v
  } else {
    if(key==k) {
      val <- c(val,v)
    } else{
      key <- k
      val <- v
    }
  }
# emitting key and list of values
  keyval(key,list(val))
}
```



Before executing the MapReduce job, please start all the Hadoop daemons and check the HDFS connection via the `hdfs.init()` method. If your Hadoop daemons have not been started, you can start them by `$hduser@ubuntu :~ $HADOOP_HOME/bin/start-all.sh`.

Once we are ready with the logic of the Mapper and Reducer, MapReduce jobs can be executed by the MapReduce method of the `rnr2` package. Here we have developed multiple MapReduce jobs, so we need to call the `mapreduce` function within the `mapreduce` function with the required parameters.

The command for calling a chained MapReduce job is seen in the following figure:

```
# executing Hadoop MapReduce
output <- mapreduce(input=mapreduce(input=webpages.values,
                                     map = mapper1,
                                     reduce = reducer1),
                    map = mapper2,
                    reduce = reducer2,
                    combine = TRUE)
```

The following is the command for retrieving the generated output from HDFS:

```
from.dfs(output)
```

While executing Hadoop MapReduce, the execution log output will be printed over the terminal for the purpose of monitoring. We will understand MapReduce job 1 and MapReduce job 2 by separating them into different parts.

The details for MapReduce job 1 is as follows:

- **Tracking the MapReduce job metadata:** With this initial portion of log, we can identify the metadata for the Hadoop MapReduce job. We can also track the job status with the web browser by calling the given Tracking URL.

```
packageJobJar: [/tmp/Rtmpn7tKAv/rmr-local-enve056d43a14e, /tmp/Rtmpn7tKAv/rmr-global-enve05d2d95c,
/tmp/Rtmpn7tKAv/rmr-streaming-mape055f73cd8c, /tmp/Rtmpn7tKAv/rmr-streaming-reducee0529eb924e, /app/hadoop/tmp/hadoop-
unjar6101512165582043075/] [] /tmp/streamjob5358198509362391792.jar tmpDir=null
13/10/24 13:45:21 INFO mapred.FileInputFormat: Total input paths to process : 1
13/10/24 13:45:22 INFO streaming.StreamJob: getLocalDirs(): [/app/hadoop/tmp/mapred/local]
13/10/24 13:45:22 INFO streaming.StreamJob: Running job: job_201310241342_0001
13/10/24 13:45:22 INFO streaming.StreamJob: To kill this job, run:
13/10/24 13:45:22 INFO streaming.StreamJob: /usr/local/hadoop/libexec/bin/hadoop job -
Dmapred.job.tracker=localhost:54311 -kill job_201310241342_0001
13/10/24 13:45:22 INFO streaming.StreamJob: Tracking URL: http://localhost:50030/jobdetails.jsp?
jobid=job_201310241342_0001
```

- **Tracking status of Mapper and Reducer tasks:** With this portion of log, we can monitor the status of the Mapper or Reducer task being run on Hadoop cluster to get details such as whether it was a success or a failure.

```
13/10/24 13:45:23 INFO streaming.StreamJob: map 0% reduce 0%
13/10/24 13:46:02 INFO streaming.StreamJob: map 50% reduce 0%
13/10/24 13:46:11 INFO streaming.StreamJob: map 100% reduce 0%
13/10/24 13:46:20 INFO streaming.StreamJob: map 100% reduce 17%
13/10/24 13:46:23 INFO streaming.StreamJob: map 100% reduce 100%
13/10/24 13:46:35 INFO streaming.StreamJob: Job complete: job_201310241342_0001
```

- **Tracking HDFS output location:** Once the MapReduce job is completed, its output location will be displayed at the end of logs.

```
13/10/24 13:46:35 INFO streaming.StreamJob: Output: /tmp/Rtmpn7tKAv/filee05467211b
```

For MapReduce job 2.

- **Tracking the MapReduce job metadata:** With this initial portion of log, we can identify the metadata for the Hadoop MapReduce job. We can also track the job status with the web browser by calling the given Tracking URL.

```
packageJobJar: [/tmp/Rtmpn7tKAv/rmr-local-enve055fb43a38, /tmp/Rtmpn7tKAv/rmr-global-enve0549c1f8a5,
/tmp/Rtmpn7tKAv/rmr-streaming-mape052bf9ab69, /tmp/Rtmpn7tKAv/rmr-streaming-reducee0552cf3c79, /tmp/Rtmpn7tKAv/rmr-
streaming-combinee0534783636, /app/hadoop/tmp/hadoop-unjar2866287784631685861/] []
/tmp/streamjob8204691495163848860.jar tmpDir=null
13/10/24 13:46:38 INFO mapred.FileInputFormat: Total input paths to process : 1
13/10/24 13:46:38 INFO streaming.StreamJob: getLocalDirs(): [/app/hadoop/tmp/mapred/local]
13/10/24 13:46:38 INFO streaming.StreamJob: Running job: job_201310241342_0002
13/10/24 13:46:38 INFO streaming.StreamJob: To kill this job, run:
13/10/24 13:46:38 INFO streaming.StreamJob: /usr/local/hadoop/libexec/bin/hadoop job -
Dmapred.job.tracker=localhost:54311 -kill job_201310241342_0002
13/10/24 13:46:38 INFO streaming.StreamJob: Tracking URL: http://localhost:50030/jobdetails.jsp?
jobid=job_201310241342_0002
```

- **Tracking status of the Mapper and Reducer tasks:** With this portion of log, we can monitor the status of the Mapper or Reducer tasks being run on the Hadoop cluster to get the details such as whether it was successful or failed.

```
13/10/24 13:46:39 INFO streaming.StreamJob: map 0% reduce 0%
13/10/24 13:46:56 INFO streaming.StreamJob: map 4% reduce 0%
13/10/24 13:46:59 INFO streaming.StreamJob: map 9% reduce 0%
13/10/24 13:47:08 INFO streaming.StreamJob: map 55% reduce 0%
13/10/24 13:47:15 INFO streaming.StreamJob: map 100% reduce 0%
13/10/24 13:47:30 INFO streaming.StreamJob: map 100% reduce 33%
13/10/24 13:47:36 INFO streaming.StreamJob: map 100% reduce 100%
13/10/24 13:47:42 INFO streaming.StreamJob: Job complete: job_201310241342_0002
```

- **Tracking HDFS output location:** Once the MapReduce job is completed, its output location will be displayed at the end of the logs.

```
13/10/24 13:47:42 INFO streaming.StreamJob: Output: /tmp/Rtmpn7tKAv/filee05767f2
```

The output of this chained MapReduce job is stored at an HDFS location, which can be retrieved by the command:

```
from.dfs(output)
```

The response to the preceding command is shown in the following figure (output only for the top 1000 rows of the dataset):

```
$key
[1] "low" "high"

$val
$val[[1]]
$val[[1]][[1]]
                                pagePath
1  /medicalcollege/s-s-agarwal-college-of-nursing-navsari
2                                /meritcalc.php
3                                /merit-calculator
4                                /new_mba_college_list.php
5                                /newmecollege.php
6                                /newmpharmcollege.php
7                                /ourteam.php
8                                /search
9                                /search2.php
10                               /search3.php
11                               /search.php

$val[[1]][[2]]
                                pagePath
1                                /abbranch.php
2                                /admission/
3                                /admission/
4  /admission/diplomaadmissionpossibilities1.php
5                                /advertisewithus
6                                /bindingcollegedetails.php
7                                /bindingmpharmcollege.php
8                                /engineering-admission-possibilities
9                                /gtuadmissionhelpline-team
10                               /gujarat-degree-engineering-college
11  /MBA-MCA/aes-institute-of-computer-studies

$val[[2]]
$val[[2]][[1]]
                                pagePath
12                               /
--                               ,
```

Visualizing data

We collected the web page categorization output using the three categories. I think the best thing we can do is simply list the URLs. But if we have more information, such as sources, we can represent the web pages as nodes of a graph, colored by popularity with directed edges when users follow the links. This can lead to more informative insights.

Computing the frequency of stock market change

This data analytics MapReduce problem is designed for calculating the frequency of stock market changes.

Identifying the problem

Since this is a typical stock market data analytics problem, it will calculate the frequency of past changes for one particular symbol of the stock market, such as a **Fourier Transformation**. Based on this information, the investor can get more insights on changes for different time periods. So the goal of this analytics is to calculate the frequencies of percentage change.

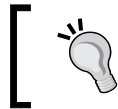
Yahoo finance data for symbol BP						
Date	Open	High	Low	Close	Volume	Adj Close
2013-08-23	41.16	41.54	41.11	41.51	4117400	41.51
2013-08-22	40.82	40.99	40.75	40.91	2808300	40.91
2013-08-21	40.84	40.89	40.51	40.53	4296800	40.53
2013-08-20	41.02	40.90	40.90	4354200	40.90	
2013-08-19	41.29	41.35	41.05	41.10	3633800	41.10

Change frequency calculation for Yahoo Finance data						
Change	Frequency					
-0.1	20					
0.3	2					
0.8	1					
1.0	22					
1.9	12					

Designing data requirement

For this stock market analytics, we will use Yahoo! Finance as the input dataset. We need to retrieve the specific symbol's stock information. To retrieve this data, we will use the Yahoo! API with the following parameters:

- From month
- From day
- From year
- To month
- To day
- To year
- Symbol



For more information on this API, visit
<http://developer.yahoo.com/finance/>.

Preprocessing data

To perform the analytics over the extracted dataset, we will use R to fire the following command:

```
stock_BP <- read.csv("http://ichart.finance.yahoo.com/table.csv?s=BP")
```

Or you can also download via the terminal:

```
wget http://ichart.finance.yahoo.com/table.csv?s=BP
#exporting to csv file
```

```
write.csv(stock_BP, "table.csv", row.names=FALSE)
```

Then upload it to HDFS by creating a specific Hadoop directory for this:

```
# creating /stock directory in hdfs
bin/hadoop dfs -mkdir /stock
```

```
# uploading table.csv to hdfs in /stock directory
bin/hadoop dfs -put /home/Vignesh/downloads/table.csv /stock/
```

Performing analytics over data

To perform the data analytics operations, we will use streaming with R and Hadoop (without the HadoopStreaming package). So, the development of this MapReduce job can be done without any RHadoop integrated library/package.

In this MapReduce job, we have defined Map and Reduce in different R files to be provided to the Hadoop streaming function.

- **Mapper:** stock_mapper.R

```
#!/usr/bin/env/Rscript
# To disable the warnings
options(warn=-1)
# To take input the data from streaming
input <- file("stdin", "r")

# To reading the each lines of documents till the end
while(length(currentLine <-readLines(input, n=1, warn=FALSE)) > 0)
{

# To split the line by "," separator
fields <- unlist(strsplit(currentLine, ","))

# Capturing open column value
open <- as.double(fields[2])

# Capturing close columns value
close <- as.double(fields[5])

# Calculating the difference of close and open attribute
change <- (close-open)

# emitting change as key and value as 1
write(paste(change, 1, sep="\t"), stdout())
}

close(input)
```

- **Reducer:** `stock_reducer.R`

```
#!/usr/bin/env Rscript
stock.key <- NA
stock.val <- 0.0

conn <- file("stdin", open="r")
while (length(next.line <- readLines(conn, n=1)) > 0) {
  split.line <- strsplit(next.line, "\t")
  key <- split.line[[1]][1]
  val <- as.numeric(split.line[[1]][2])
  if (is.na(current.key)) {
    current.key <- key
    current.val <- val
  }
  else {
    if (current.key == key) {
      current.val <- current.val + val
    }
    else {
      write(paste(current.key, current.val, sep="\t"), stdout())
      current.key <- key
      current.val <- val
    }
  }
  write(paste(current.key, current.val, sep="\t"), stdout())
  close(conn)
```

From the following codes, we run MapReduce in R without installing or using any R library/package. There is one `system()` method in R to fire the system command within R console to help us direct the firing of Hadoop jobs within R. It will also provide the repose of the commands into the R console.

```
# For locating at Hadoop Directory
system("cd $HADOOP_HOME")

# For listing all HDFS first level directory
system("bin/hadoop dfs -ls /")

# For running Hadoop MapReduce with streaming parameters
system(paste("bin/hadoop jar
/usr/local/hadoop/contrib/streaming/hadoop-streaming-1.0.3.jar ",

"-input /stock/table.csv",
"-output /stock/outputs",
"-file /usr/local/hadoop/stock/stock_mapper.R",
"-mapper /usr/local/hadoop/stock/stock_mapper.R",
"-file /usr/local/hadoop/stock/stock_reducer.R",
"-reducer /usr/local/hadoop/stock/stock_reducer.R"))

# For storing the output of list command
dir <- system("bin/hadoop dfs -ls /stock/outputs", intern=TRUE)
dir

# For storing the output from part-0000 (output file)
out <- system("bin/hadoop dfs -cat /stock/outputs/part-00000",
intern=TRUE)

# displaying Hadoop MapReduce output data out
```

You can also run this same program via the terminal:

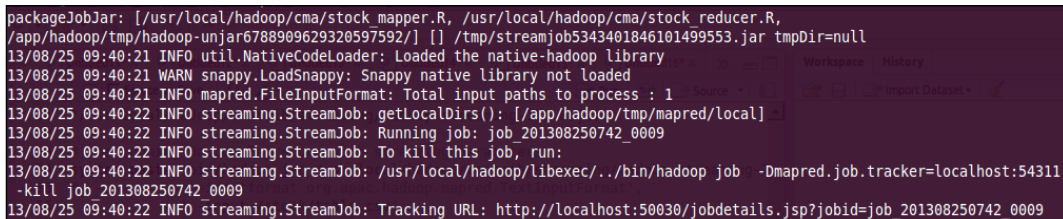
```
bin/hadoop jar /usr/local/hadoop/contrib/streaming/hadoop-streaming-
1.0.3.jar \

-input /stock/table.csv \
-output /stock/outputs\
-file /usr/local/hadoop/stock/stock_mapper.R \
-mapper /usr/local/hadoop/stock/stock_mapper.R \
-file /usr/local/hadoop/stock/stock_reducer.R \
-reducer /usr/local/hadoop/stock/stock_reducer.R
```

While running this program, the output at your R console or terminal will be as given in the following screenshot, and with the help of this we can monitor the status of the Hadoop MapReduce job. Here we will see them sequentially with the divided parts. Please note that we have separated the logs output into parts to help you understand them better.

The MapReduce log output contains (when run from terminal):

- With this initial portion of log, we can identify the metadata for the Hadoop MapReduce job. We can also track the job status with the web browser, by calling the given Tracking URL. This is how the MapReduce job metadata is tracked.



```
packageJobJar: [/usr/local/hadoop/cma/stock_mapper.R, /usr/local/hadoop/cma/stock_reducer.R,
/app/hadoop/tmp/hadoop-unjar6788909629320597592/] [] /tmp/streamjob5343401846101499553.jar tmpDir=null
13/08/25 09:40:21 INFO util.NativeCodeLoader: Loaded the native-hadoop library
13/08/25 09:40:21 WARN Snappy.LoadSnappy: Snappy native library not loaded
13/08/25 09:40:21 INFO mapred.FileInputFormat: Total input paths to process : 1
13/08/25 09:40:22 INFO streaming.StreamJob: getLocalDirs(): [/app/hadoop/tmp/mapred/local]
13/08/25 09:40:22 INFO streaming.StreamJob: Running job: job_201308250742_0009
13/08/25 09:40:22 INFO streaming.StreamJob: To kill this job, run:
13/08/25 09:40:22 INFO streaming.StreamJob: /usr/local/hadoop/libexec/bin/hadoop job -Dmapred.job.tracker=localhost:54311
-kill job_201308250742_0009
13/08/25 09:40:22 INFO streaming.StreamJob: Tracking URL: http://localhost:50030/jobdetails.jsp?jobid=job_201308250742_0009
```

- With this portion of log, we can monitor the status of the Mapper or Reducer tasks being run on the Hadoop cluster to get the details like whether it was successful or failed. This is how we track the status of the Mapper and Reducer tasks.

```
13/08/0:23 INFO streaming.StreamJob:
map 0% reduce 0%

13/08/25 09:41:39 INFO streaming.StreamJob:
map 29% reduce 0%

13/08/25 09:41:43 INFO streaming.StreamJob:
map 59% reduce 0%

13/08/25 09:41:58 INFO streaming.StreamJob:
map 79% reduce 0%

13/08/25 09:42:01 INFO streaming.StreamJob:
map 100% reduce 0%

13/08/25 09:42:44 INFO streaming.StreamJob:
map 100% reduce 100%

13/08/25 09:43:02 INFO streaming.StreamJob:
Job complete: job_201308250742_0009
```

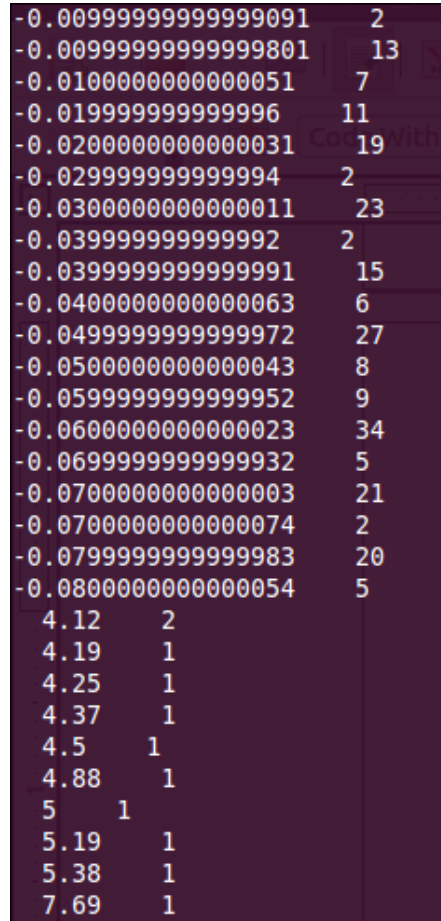
- Once the MapReduce job is completed, its output location will be displayed at the end of the logs. This is known as tracking the HDFS output location.

```
13/08/25 09:43:02 INFO streaming.StreamJob:
Output: /stock/outputs
```

- From the terminal, the output of the Hadoop MapReduce program can be called using the following command:

```
bin/hadoop dfs -cat /stock/outputs/part-00000
```

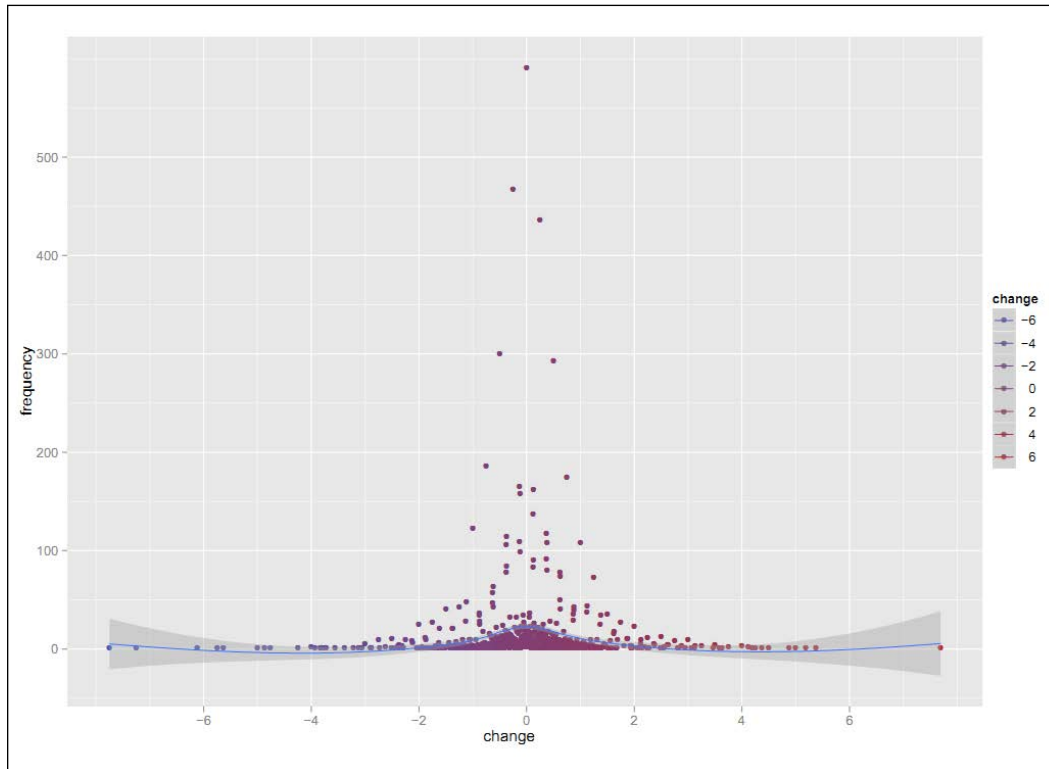
- The headers of the output of your MapReduce program will look as follows:
change frequency
- The following figure shows the sample output of MapReduce problem:



```
-0.00999999999999991 2
-0.009999999999999801 13
-0.010000000000000051 7
-0.01999999999999996 11
-0.020000000000000031 19
-0.02999999999999994 2
-0.030000000000000011 23
-0.03999999999999992 2
-0.03999999999999991 15
-0.040000000000000063 6
-0.04999999999999972 27
-0.050000000000000043 8
-0.05999999999999952 9
-0.060000000000000023 34
-0.06999999999999932 5
-0.070000000000000003 21
-0.070000000000000074 2
-0.07999999999999983 20
-0.080000000000000054 5
4.12 2
4.19 1
4.25 1
4.37 1
4.5 1
4.88 1
5 1
5.19 1
5.38 1
7.69 1
```

Visualizing data

We can get more insights if we visualize our output with various graphs in R. Here, we have tried to visualize the output with the help of the ggplot2 package.



From the previous graph, we can quickly identify that most of the time the stock price has changed from around 0 to 1.5. So, the stock's price movements in the history will be helpful at the time of investing.

The required code for generating this graph is as follows:

```
# Loading ggplot2 library
library(ggplot2);

# we have stored above terminal output to stock_output.txt file

#loading it to R workspace
myStockData <- read.delim("stock_output.txt", header=F, sep=" ",
dec=".");

# plotting the data with ggplot2 geom_smooth function
ggplot(myStockData, aes(x=V1, y=V2)) + geom_smooth() + geom_point();
```

In the next section, we have included the case study on how Big Data analytics is performed with R and Hadoop for the **Kaggle** data competition.

Predicting the sale price of blue book for bulldozers – case study

This is a case study for predicting the auction sale price for a piece of heavy equipment to create a blue book for bulldozers.

Identifying the problem

In this example, I have included a case study by Cloudera data scientists on how large datasets can be resampled, and applied the random forest model with R and Hadoop. Here, I have considered the Kaggle blue book for bulldozers competition for understanding the types of Big Data problem definitions. Here, the goal of this competition is to predict the sale price of a particular piece of heavy equipment at a usage auction based on its usage, equipment type, and configuration. This solution has been provided by *Uri Laserson* (Data Scientist at Cloudera). The provided data contains the information about auction result posting, usage, and equipment configuration.

It's a trick to model the Big Data sets and divide them into the smaller datasets. Fitting the model on that dataset is a traditional machine learning technique such as random forests or bagging. There are possibly two reasons for random forests:

- Large datasets typically live in a cluster, so any operations will have some level of parallelism. Separate models fit on separate nodes that contain different subsets of the initial data.
- Even if you can use the entire initial dataset to fit a single model, it turns out that ensemble methods, where you fit multiple smaller models by using subsets of data, generally outperform single models. Indeed, fitting a single model with 100M data points can perform worse than fitting just a few models with 10M data points each (so smaller total data outperforms larger total data).

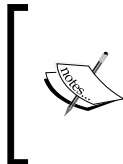
Sampling with replacement is the most popular method for sampling from the initial dataset for producing a collection of samples for model fitting. This method is equivalent to sampling from a multinomial distribution, where the probability of selecting any individual input data point is uniform over the entire dataset.



Kaggle is a Big Data platform where data scientists from all over the world compete to solve Big Data analytics problems hosted by data-driven organizations.

Designing data requirement

For this competition, Kaggle has provided real-world datasets that comprises approximately 4,00,000 training data points. Each data point represents the various attributes of sales, configuration of the bulldozer, and sale price. To find out where to predict the sales price, the random forest regression model needs to be implemented.

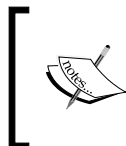


The reference link for this Kaggle competition is <http://www.kaggle.com/c/bluebook-for-bulldozers>. You can check the data, information, forum, and leaderboard as well as explore some other Big Data analytics competitions and participate in them to evaluate your data analytics skills.

We chose this model because we are interested in predicting the sales price in numeric values from random sets of a large dataset.

The datasets are provided in the terms of the following data files:

File name	Description format (size)
Train	This is a training set that contains data for 2011.
Valid	This is a validation set that contains data from January 1, 2012 to April 30, 2012.
Data dictionary	This is the metadata of the training dataset variables.
Machine_Appendix	This contains the correct year of manufacturing for a given machine along with the make, model, and product class details.
Test	This tests datasets.
random_forest_benchmark_test	This is the benchmark solution provided by the host.



In case you want to learn and practice Big Data analytics, you can acquire the Big Data sets from the Kaggle data source by participating in the Kaggle data competitions. These contain the datasets of various fields from industries worldwide.

Preprocessing data

To perform the analytics over the provided Kaggle datasets, we need to build a predictive model. To predict the sale price for the auction, we will fit the model over provided datasets. But the datasets are provided with more than one file. So we will merge them as well as perform data augmentation for acquiring more meaningful data. We are going to build a model from `Train.csv` and `Machine_Appendix.csv` for better prediction of the sale price.

Here are the data preprocessing tasks that need to be performed over the datasets:

```
# Loading Train.csv dataset which includes the Sales as well as
machine identifier data attributes.

transactions <- read.table(file=~ /Downloads/Train.csv",
header=TRUE,
sep=",",
quote="\\"",
row.names=1,
fill=TRUE,
colClasses=c(MachineID="factor",
  ModelID="factor",
  datasource="factor",
  YearMade="character",
  SalesID="character",
  auctioneerID="factor",
  UsageBand="factor",
  saledate="custom.date.2",
  Tire_Size="tire.size",
  Undercarriage_Pad_Width="undercarriage",
  Stick_Length="stick.length"),
na.strings=na.values)

# Loading Machine_Appendix.csv for machine configuration information

machines <- read.table(file=~ /Downloads/Machine_Appendix.csv",
header=TRUE,
sep=",",
quote="\\"",
fill=TRUE,
colClasses=c(MachineID="character",
  ModelID="factor",
  fiManufacturerID="factor"),
na.strings=na.values)
```

```
# Updating the values to numeric
# updating sale data number
transactions$saleDateNumeric <- as.numeric(transactions$saleDate)
transactions$ageAtSale <- as.numeric(transactions$saleDate -
as.Date(transactions$YearMade, format="%Y"))

transactions$saleYear <- as.numeric(format(transactions$saleDate,
"%Y"))

# updating the month of sale from transaction
transactions$saleMonth <- as.factor(format(transactions$saleDate,
"%B"))

# updating the date of sale from transaction
transactions$saleDay <- as.factor(format(transactions$saleDate, "%d"))

# updating the day of week of sale from transaction
transactions$saleWeekday <- as.factor(format(transactions$saleDate,
"%A"))

# updating the year of sale from transaction
transactions$YearMade <- as.integer(transactions$YearMade)

# deriving the model price from transaction
transactions$MedianModelPrice <- unsplit(lapply(split(transactions$SalePrice,
transactions$ModelID), median), transactions$ModelID)

# deriving the model count from transaction
transactions$ModelCount <- unsplit(lapply(split(transactions$SalePrice,
transactions$ModelID), length), transactions$ModelID)

# Merging the transaction and machine data in to dataframe
training.data <- merge(x=transactions, y=machines, by="MachineID")

# write denormalized data out
write.table(x=training.data,
file=~ /temp/training.csv",
sep=",",
quote=TRUE,
row.names=FALSE,
eol="\n",
col.names=FALSE)
# Create poisson directory at HDFS
bin/hadoop dfs -mkdir /poisson

# Uploading file training.csv at HDFS
bin/hadoop dfs -put ~/temp/training.csv /poisson/
```

Performing analytics over data

As we are going to perform analytics with sampled datasets, we need to understand how many datasets need to be sampled.

For random sampling, we have considered three model parameters, which are as follows:

- We have N data points in our initial training set. This is very large (10^6 - 10^9) and is distributed over an HDFS cluster.
- We are going to train a set of M different models for an ensemble classifier.
- Each of the M models will be fitted with K data points, where typically $K \ll N$. (For example, K may be 1-10 percent of N .)

We have N numbers of training datasets, which are fixed and generally outside our control. As we are going to handle this via **Poisson** sampling, we need to define the total number of input vectors to be consumed into the random forest model.

There are three cases to be considered:

- **$KM < N$** : In this case, we are not using the full amount of data available to us
- **$KM = N$** : In this case, we can exactly partition our dataset to produce totally independent samples
- **$KM > N$** : In this case, we must resample some of our data with replacements

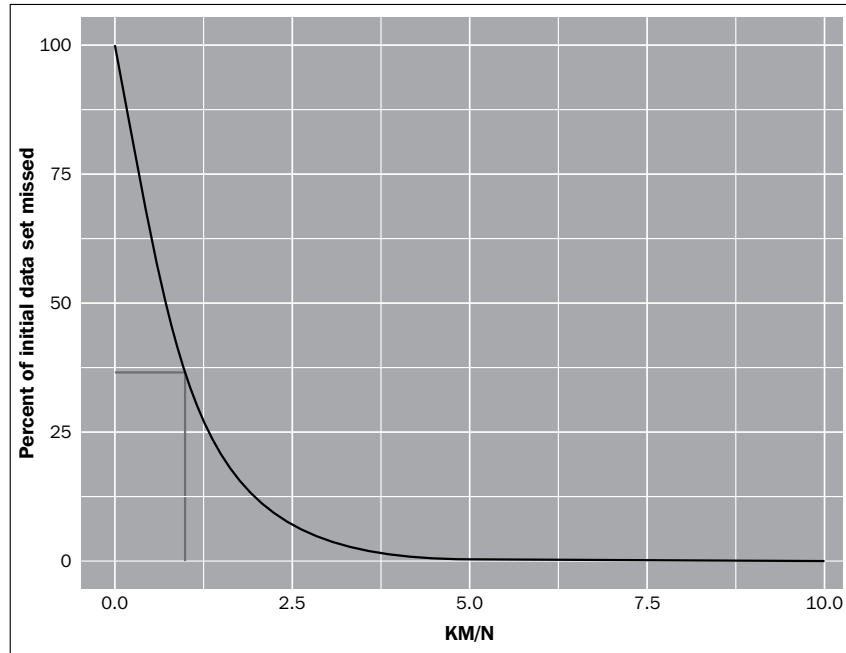
The Poisson sampling method described in the following section handles all the three cases in the same framework. However, note that for the case $KM = N$, it does not partition the data, but simply resamples it.

Understanding Poisson-approximation resampling

Generalized linear models are an extension of the general linear model. Poisson regression is a situation of generalized models. The dependent variable obeys Poisson distribution.

Poisson sampling will be run on the Map of the MapReduce task because it occurs for input data points. This doesn't guarantee that every data point will be considered into the model, which is better than multinomial resampling of full datasets. But it will guarantee the generation of independent samples by using N training input points.

Here, the following graph indicates the amount of missed datasets that can be retrieved in the Poisson sampling with the function of KM/N :



The grey line indicates the value of $KM=N$. Now, let's look at the pseudo code of the MapReduce algorithm. We have used three parameters: N , M , and K where K is fixed. We used $T=K/N$ to eliminate the need for the value of N in advance.

- **An example of sampling parameters:** Here, we will implement the preceding logic with a pseudo code. We will start by defining two model input parameters as `frac.per.model` and `num.models`, where `frac.per.model` is used for defining the fraction of the full dataset that can be used, and `num.models` is used for defining how many models will be fitted from the dataset.

```
T = 0.1 # param 1: K / N-average fraction of input data in each  
model 10%
```

```
M = 50 # param 2: number of models
```

- **Logic of Mapper:** Mapper will be designed for generating the samples of the full dataset by data wrangling.

```
def map(k, v):
    // for each input data point
    for i in 1:M
        // for each model
        m = Poisson(T)
        // num times curr point should appear in this sample
        if m > 0
            for j in 1:m
                // emit current input point proper num of times
                emit (i, v)
```

- **Logic of Reducer:** Reducer will take a data sample as input and fit the random forest model over it.

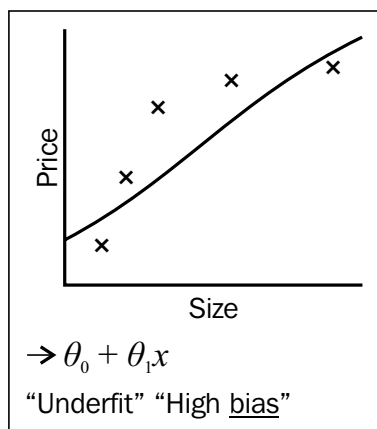
```
def reduce(k, v):
    fit model or calculate statistic with the sample in v
```

Fitting random forests with RHadoop

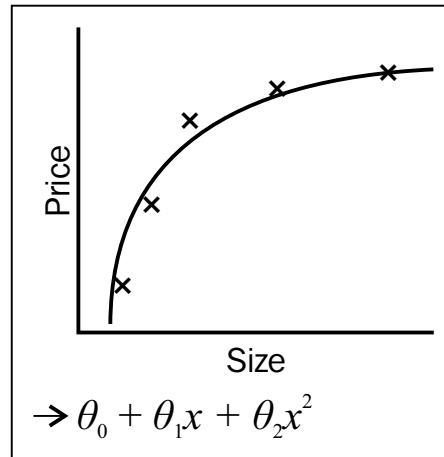
In machine learning, fitting a model means fitting the best line into our data. Fitting a model can fall under several types, namely, under fitting, over fitting, and normal fitting. In case of under and over fitting, there are chances of high bias (cross validation and training errors are high) and high variance (cross validation error is high but training error is low) effects, which is not good. We will normally fit the model over the datasets.

Here are the diagrams for fitting a model over datasets with three types of fitting:

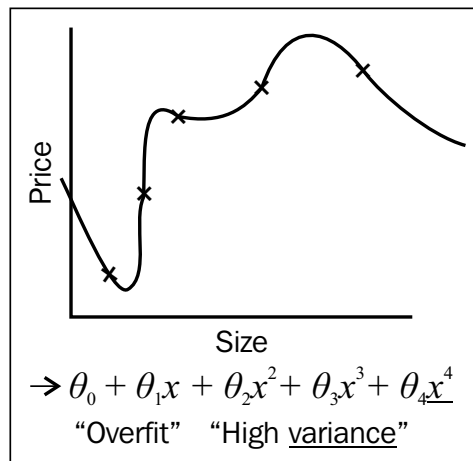
- **Under fitting:** In this cross validation and training errors are high



- **Normal fitting:** In this cross-validation and training errors are normal



- **Over fitting:** In this the cross-validation error is high but training error is low



We will fit the model over the data using the random forest technique of machine learning. This is a type of recursive partitioning method, particularly well suited for small and large problems. It involves an ensemble (or set) of classification (or regression) trees that are calculated on random subsets of the data, using a subset of randomly restricted and selected predictors for every split in each classification tree.

Furthermore, the results of an ensemble of classification/regression trees have been used to produce better predictions instead of using the results of just one classification tree.

We will now implement our Poisson sampling strategy with RHadoop. We will start by setting global values for our parameters:

```
#10% of input data to each sample on avg
frac.per.model <- 0.1
num.models <- 50
```

Let's check how to implement Mapper as per the specifications in the pseudo code with RHadoop.

- Mapper is implemented in the the following manner:

```
poisson.subsample <- function(k, input) {
  # this function is used to generate a sample from the current
  block of data
  generate.sample <- function(i) {
    # generate N Poisson variables
    draws <- rpois(n=nrow(input), lambda=frac.per.model)
    # compute the index vector for the corresponding rows,
    # weighted by the number of Poisson draws
    indices <- rep((1:nrow(input)), draws)
    # emit the rows; RHadoop takes care of replicating the key
    appropriately
    # and rbinding the data frames from different mappers together
    for the
    # reducer
    keyval(i, input[indices, ])
  }

  # here is where we generate the actual sampled data
  c.keyval(lapply(1:num.models, generate.sample))
}
```

Since we are using R, it's tricky to fit the model with the random forest model over the collected sample dataset.

- Reducer is implemented in the following manner:


```
# REDUCE function
fit.trees <- function(k, v) {
  # rmr rbinds the emitted values, so v is a dataframe
  # note that do.trace=T is used to produce output to stderr to keep
  the reduce task from timing out
  rf <- randomForest(formula=model.formula,
                     data=v,
                     na.action=na.roughfix,
                     ntree=10,
                     do.trace=FALSE)

  # rf is a list so wrap it in another list to ensure that only
  # one object gets emitted. this is because keyval is vectorized
  keyval(k, list(forest=rf))
}
```

- To fit the model, we need `model.formula`, which is as follows:

```
model.formula <- SalePrice ~ datasource + auctioneerID + YearMade
+ saledatenumeric + ProductSize + ProductGroupDesc.x + Enclosure
+ Hydraulics + ageAtSale + saleYear + saleMonth + saleDay +
saleWeekday + MedianModelPrice + ModelCount + MfgYear
```

`SalePrice` is defined as a response variable and the rest of them are defined as predictor variables for the random forest model.

[ Random forest model with R doesn't support factor with level more than 32.]

- The MapReduce job can be executed using the following command:

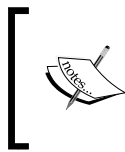
```
mapreduce(input="/poisson/training.csv",
          input.format=bulldozer.input.format,
          map=poisson.subsample,
          reduce=fit.trees,
          output="/poisson/output")
```

The resulting trees are dumped in HDFS at `/poisson/output`.

- Finally, we can load the trees, merge them, and use them to classify new test points:

```
mraw.forests <- values(from.dfs("/poisson/output"))  
forest <- do.call(combine, raw.forests)
```

Each of the 50 samples produced a random forest with 10 trees, so the final random forest is a collection of 500 trees, fitted in a distributed fashion over a Hadoop cluster.



The full set of source files is available on the official Cloudera blog at <http://blog.cloudera.com/blog/2013/02/how-to-resample-from-a-large-data-set-in-parallel-with-r-on-hadoop/>.

Hopefully, we have learned a scalable approach for training ensemble classifiers or bootstrapping in a parallel fashion by using a Poisson approximation for multinomial sampling.

Summary

In this chapter, we learned how to perform Big Data analytics with various data driven activities over an R and Hadoop integrated environment.

In the next chapter, we will learn more about how R and Hadoop can be used to perform machine learning techniques.