National Institute of Technology
Rourkela, India

Big Data Analysis of Generic Medicines in Pharmaceutical Records

An implementation in Java using MapReduce technique in Hadoop

Adyasha Panda
Department of Computer Science & Engineering
IIIT Bhubaneswar
Email: adyashapanda@gmail.com

Mentor
Dr. Bansidhar Majhi
Professor, Department of Computer Science & Engineering
National Institute of Technology Rourkela
Email: bmajhi@nitrkl.ac.in

Last updated on: 6 January, 2015
Cite as: IIITBH-ACM/TR/2015/001
Permalink: https://raw.githubusercontent.com/iiitbh-acm/TR/master/2015/IIITBH-ACM_TR_2015_001.pdf

This work was partially supported by National Institute of Technology (NIT), Rourkela during the period December 2014. The author would like to thank Dr. Bansidhar Majhi for his guidance.

Department of Computer Science & Engineering
International Institute of Information Technology
Bhubaneswar, India

ABSTRACT

Reducing the Cost of Care. Payers, whether governments or private insurers, face a huge hurdle in bending the cost curve downward to slow the pace of growth in health care expenses. One area ripe for improvement lies in reducing the cost of care. Since the cost of care generally accounts for 90 to 95 percent of total costs for an efficient payer, every 1 percent reduction in the cost of care has the same effect as a 10 to 20 percent reduction in operational costs.

In the area of procuring care alone, we see enormous potential to drive down costs through the use of big data. Our target is prescription drugs, which accounts for about 15 percent of costs. We focus on prescriptions for generic drugs as a substitute for brand-name drugs. Often, generics cost less than 10 percent of branded medicines. An analysis showed that switching almost entirely to generics for just one cholesterol-controlling drug, Lipitor, would save more than 3%.

In most countries, pharmacies are obliged to deliver a generic drug instead of a branded drug. But prescribers can state that medical necessity requires the patient to receive the expensive branded drug instead. Since the active ingredient in generics is the same as in branded drugs, prescriptions for a branded drug on the grounds of medical necessity should be rare—for example, less than 5 percent of prescriptions, according to calculations based on best practices. In an effort to bring up the rate of generic adoption among doctors much more quickly, we decide to use records to pinpoint exactly who appear to be overprescribing branded drugs. [1]

I. DATASET

The dataset is a xlsx file containing the medical records of a local Hospital in Rourkela. [2] The records consisted of Visit Number, Date, Medical Card Number, Doctor ID, Complaint and Medicine Name. It was in the following format.

Visit Number	**Date**	**Medical Card Number**	**Doctor ID**	**Complaint**	**Medicine Name**
107	21-10-2013	D28115	2121058	polyarthralgia	Oace sptab
107	21-10-2013	D28115	2121058	polyarthralgia	Ramil dsr tab
108	21-10-2013	S292675	2121058	oral ulcer,muscle pain	Becosules z caps
108	21-10-2013	S292675	2121058	oral ulcer,muscle pain	Oace mrtab
108	21-10-2013	S292675	2121058	oral ulcer,muscle pain	Infladol gel 30gm

II. CONVERTING THE DATA IN TEXT FORMAT

In order to input the data into hadoop distributed file system, it is easier if we convert the data to text format to avoid using complex tools like Sqoop.

Python code for converting the dataset to text format :

```
import csv
with open('/home/ady/copy of medicaldata.csv', 'rb') as f:
        reader = csv.reader(f)
        for row in reader:
                print row
```

The output of the code is in the following format [3]:

```
['', '2121058', 'Infladol gel 30gm']
['', '2131105', 'Zyloric100 mg tab']
['', '2131105', 'Calawinlotion']
['', '2131105', 'Oa furo tab']
['', '2131105', 'Nascold tab']
['', '2131105', 'Ruby dsr tab']
['', '2131105', 'X TER OINTMENT']
```

III. MAP REDUCE

MapReduce is a programming model of a big data processing tool, Hadoop, used to process large datasets in a parallel distributed programming approach. It consists of two phases: Map and Reduce.

In the mapping phase the whole dataset is filtered and sorted, and in the Reduce phase the result is summed up. In the Map and Reduce functions the data is structured in (key,value) pairs. The Map function accepts input in the form of a value associated with each key and returns a list of values associated with every key .

```
Map(k1,v1) ->  list(k2,v2)
```

The Map function is applied in parallel to the various segments of the dataset and the result from each Map function is then collected, sorted, shuffled and fed into the Reduce function .

The reduce function accepts the output from the Map function, sums up the list of values associated with a key according to the problem statement and returns a value associated with each key.

```
Reduce [ list(k2,v2) ] -> (k3,v3)
```

IV. FINDING THE NUMBER OF PRESCRIPTIONS FOR EACH DOCTOR IN MAPREDUCE

A. MapReduce Code

```
package org.myorg;
import java.io.IOException;
import java.util.*;
import java.net.*;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.util.*;
public class pres_count
{
    public static class Map extends MapReduceBase implements Mapper<LongWritable, Text, Text, 
IntWritable>
    {
        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();
        public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output, 
Reporter reporter) throws IOException
        {
            String[] line = value.toString().split(",");
            StringTokenizer tokenizer = new StringTokenizer(line[1]);
            while (tokenizer.hasMoreTokens())
            {
                word.set(tokenizer.nextToken());
                output.collect(word, one);
            }
        }
    }
    public static class Reduce extends MapReduceBase implements Reducer<Text, IntWritable, Text, 
IntWritable>
    {
        public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text, 
IntWritable> output, Reporter reporter) throws IOException
        {
            int sum = 0;
            while (values.hasNext())
            {
                sum += values.next().get();
            }
            output.collect(key, new IntWritable(sum));
        }
    }
```

```
        public static void main(String[] args) throws Exception
        {
            JobConf conf = new JobConf(pres_count.class);
            conf.setJobName("pres_count");
            conf.setOutputKeyClass(Text.class);
            conf.setOutputValueClass(IntWritable.class);
            conf.setMapperClass(Map.class);
            conf.setCombinerClass(Reduce.class);
            conf.setReducerClass(Reduce.class);
            conf.setInputFormat(TextInputFormat.class);
            conf.setOutputFormat(TextOutputFormat.class);
            FileInputFormat.setInputPaths(conf, new Path(args[0]));
            FileOutputFormat.setOutputPath(conf, new Path(args[1]));
            JobClient.runJob(conf);
        }
  }
```

B. Output

'2121034'	6698
'2121058'	18056
'2131105'	23076
'V-101'	1106
'V-103'	234
'V-104'	866
'V-105'	2435
'V-106'	1285
'V-107'	852
'V-108'	29
'V-111'	22

V. DOCTORS WHO PRESCRIBED OVERPRICED MEDICINES AT LEAST ONCE

The entire dataset is too big to be processed in a single node hadoop cluster in polynomial time. Thus only first 1000 records of the dataset is considered for analysis in this section. Healthkartplus.com is the website used in this section, to find medicines which are generic drugs and which have substitutes, whose composition is same as the medicine prescribed by the doctor but can be considerably cheaper. [4]

A. Finding the medicines which have cheaper substitutes, and eliminating the generic medicines

```
import java.net.*;
import java.io.*;
import org.jsoup.nodes.Document;
import org.jsoup.Jsoup;
import org.jsoup.nodes.Element;
import org.jsoup.select.Elements;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.util.*;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;

public class retrieval
```

```
{
    public static class Map extends MapReduceBase implements Mapper<Object, Object, Object ,
Object >
    {
        public void map(Object arg0, Object arg1, OutputCollector arg2, Reporter arg3) throws
IOException
        {
            String[] line = arg1.toString().split(",");
            String[] words =line[2].split(" ");
            String searchName="https://www.healthkartplus.com/search/all?name=";
            for (String word : words)
            {
                searchName=searchName+word+"+";
            }

            String searchNm=searchName.substring(0,searchName.length()-2);
            int k=1;
            while(k==1)
            {
                try
                {
                    URL myURL = new URL(searchNm);
                    URLConnection myURLConnection = myURL.openConnection();
                    myURLConnection.connect();
                    if(myURL.openStream()==null)
                        continue;
                    BufferedReader in =new BufferedReader(new
InputStreamReader(myURL.openStream()));

                    String HTMLSTring = in.readLine();
                    Document doc = Jsoup.parse(HTMLSTring);
                    Elements link = doc.select("a[class=item-name prdctNm]").eq(0);

                    String text = doc.body().text();
                    String linkHref = link.attr("href");
                    String name="https://www.healthkartplus.com" + linkHref;
                    URL myURL1 = new URL(name);
                    URLConnection myURLConnection1 = myURL1.openConnection();
                    myURLConnection1.connect();
                    if(myURL1.openStream()==null)
                        continue;
                    k++;
                    BufferedReader ini =new BufferedReader(new
InputStreamReader(myURL1.openStream()));

                    String line1 = null,HTMLstring=null;
                    int i=1;
                    while (( line1 = ini.readLine()) != null)
                    {
                        if(line1.contains("SUBSTITUTES"))
                        {
                            arg2.collect(new Text(line[1]),new Text(line[2]));
                        }
                        i++;
                    }
                }

                catch(MalformedURLException e){}
                catch(IOException e){}
                catch(ArrayIndexOutOfBoundsException e){}
            }
        }
    }

    public static void main(String[] args) throws Exception
    {
        JobConf conf = new JobConf(retrieval.class);
        Job job = new Job(conf, "Map-Only Job");
        job.setJarByClass(retrieval.class);
        job.setNumReduceTasks(0);

        conf.setOutputKeyClass(Text.class);
        conf.setOutputValueClass(Text.class);
        conf.setMapperClass(Map.class);
        conf.setInputFormat(TextInputFormat.class);
        conf.setOutputFormat(TextOutputFormat.class);
        FileInputFormat.setInputPaths(conf, new Path(args[0]));
        FileOutputFormat.setOutputPath(conf, new Path(args[1]));
        JobClient.runJob(conf);
    }
}
```

Only 154 records of the 1000 records are retrieved, i.e 846 records contain generic medicines.

B. Finding the doctors who prescribed medicines having a cheaper substitute at least once

```
import java.io.IOException;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.*;
import java.util.*;
import java.net.*;
import org.jsoup.nodes.Document;
import org.jsoup.Jsoup;
import org.jsoup.nodes.Element;
import org.jsoup.select.Elements;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.util.*;

public class nitproject
{
    private final static IntWritable one = new IntWritable(1);
    private final static IntWritable zero = new IntWritable(0);
    public static class Map extends MapReduceBase implements Mapper<LongWritable, Text, Text,
IntWritable>
    {
        private Text word = new Text();
        public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output,
Reporter reporter) throws IOException
        {
            String[] line=value.toString().split("\t");
            String[] words =line[1].split(" ");
            String searchName="https://www.healthkartplus.com/search/all?name=";
            for (String word : words)
            {
                searchName=searchName+word+"+"; }
                String searchNm=searchName.substring(0,searchName.length()2);
                try
                {
                    URL myURL = new URL(searchNm);
                    URLConnection myURLConnection = myURL.openConnection();
                    myURLConnection.connect();
                    BufferedReader in =new BufferedReader(new
InputStreamReader(myURL.openStream()));
                    String HTMLSTring = in.readLine();
                    Document doc = Jsoup.parse(HTMLSTring);
                    Elements link = doc.select("a[class=itemname prdctNm]").eq(0);
                    String text = doc.body().text();
                    String linkHref = link.attr("href");
                    String name="https://www.healthkartplus.com" + linkHref;
                    URL myURL1 = new URL(name);
                    URLConnection myURLConnection1 = myURL1.openConnection();
                    myURLConnection1.connect();
                    BufferedReader ini =new BufferedReader(new
InputStreamReader(myURL1.openStream()));
                    String line1 = null,HTMLstring=null;
                    int i=1;
                    while (( line1 = ini.readLine()) != null)
                    {
                        if(line1.contains("SUBSTITUTES"))
                        {
                            HTMLstring =line1;
                        }
                        i++;
                    }
                    Document doc1 = Jsoup.parse(HTMLstring);
                    Elements match =doc1.getElementsByClass("itemsave");
                    String save=match.text();
                    String arr[] = save.split(" ", 2);
                    String firstWord = arr[0];
                    if(firstWord.equals("save"))
                    {
                        output.collect(new Text(line[0]),one);
                    }
                    else
                    {
                        output.collect(new Text(line[0]),zero);
                    }
                }
                catch(MalformedURLException e){}
                catch(IOException e){}
            }
        }
        public static class Reduce extends MapReduceBase implements Reducer<Text, IntWritable,
Text, IntWritable>
        {
            public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text,
IntWritable> output, Reporter reporter) throws IOException
            {
                int flag = 0;
                while (values.hasNext())
                {
                    if(values.next().get() == 1)
                    {
                        flag = 1;
                    }
                }
                if( flag == 1)
                {
                    output.collect(key, one);
                }
                else
                {
                    output.collect(key, zero);
                }
            }
        }
        public static void main(String[] args) throws Exception
        {
            JobConf conf = new JobConf(nitproject.class);
            conf.setJobName("nitproject");
            conf.setOutputKeyClass(Text.class); conf.setOutputValueClass(IntWritable.class);
            conf.setMapperClass(Map.class);
            conf.setCombinerClass(Reduce.class);
            conf.setReducerClass(Reduce.class);
            conf.setInputFormat(TextInputFormat.class);
            conf.setOutputFormat(TextOutputFormat.class);
            FileInputFormat.setInputPaths(conf, new Path(args[0]));
            FileOutputFormat.setOutputPath(conf, new Path(args[1]));
             JobClient.runJob(conf);
         }
}
```

C. Result

```
'2121058'      1
'2131105'      1
'V101'         0
'V105'         1
'V106'         1
```

80% of the doctors have prescribed an overpriced medicine at least once.

D. Finding the number of doctors who prescribed overpriced medicines greater than or equal to 25% of the time

```java
import java.io.IOException;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.*;
import java.util.*;
import java.net.*;
import org.jsoup.nodes.Document;
import org.jsoup.Jsoup;
import org.jsoup.nodes.Element;
import org.jsoup.select.Elements;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.util.*;

public class nitproj25
{
    private final static IntWritable one = new IntWritable(1);
    private final static IntWritable zero = new IntWritable(0);
    public static class Map extends MapReduceBase implements Mapper<LongWritable, Text, Text,
IntWritable>
    {
        private Text word = new Text();
        public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output,
Reporter reporter) throws IOException
        {
            String[] line=value.toString().split("\t");
            String[] words =line[1].split(" ");
            String searchName="https://www.healthkartplus.com/search/all?name=";
            for (String word : words)
            {
                searchName=searchName+word+"+";
            }
            String searchNm=searchName.substring(0,searchName.length()2);
            try
            {
                URL myURL = new URL(searchNm);
                URLConnection myURLConnection = myURL.openConnection();
                myURLConnection.connect();
                BufferedReader in =new BufferedReader(new InputStreamReader(myURL.openStream()));

                String HTMLSTring = in.readLine();
                Document doc = Jsoup.parse(HTMLSTring);
                Elements link = doc.select("a[class=itemname prdctNm]").eq(0);
                String text = doc.body().text();
                String linkHref = link.attr("href");
                String name="https://www.healthkartplus.com" + linkHref;
                URL myURL1 = new URL(name);
                URLConnection myURLConnection1 = myURL1.openConnection();
                myURLConnection1.connect();
                BufferedReader ini =new BufferedReader(new
InputStreamReader(myURL1.openStream()));
                String line1 = null,HTMLstring=null;
                int i=1;
                while (( line1 = ini.readLine()) != null)
                {
                    if(line1.contains("SUBSTITUTES"))
                    {
                        HTMLstring =line1;
                    }
                    i++;
                }
                Document doc1 = Jsoup.parse(HTMLstring); Elements
match=doc1.getElementsByClass("itemsave");
                String save=match.text();
                String arr[] = save.split(" ", 2);
                String firstWord = arr[0];
                if(firstWord.equals("save"))
                {
                    output.collect(new Text(line[0]),one);
                }
                else
                {
                    output.collect(new Text(line[0]),zero);
                }
            }
            catch(MalformedURLException e){}
            catch(IOException e){}
        }
    }

    public static class Reduce extends MapReduceBase implements Reducer<Text, IntWritable, Text,
IntWritable>
    {
        public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text,
IntWritable> output, Reporter reporter) throws IOException
        {
            int num = 0, sum=0, percent=0;
            while (values.hasNext())
            {
                sum++;
                if(values.next().get() == 1)
                {
                    num++;
                }
            }
            percent=(num/sum)*100;
            if(percent >= 10)
            {
                output.collect(key, one);
            }
            else
            {
                output.collect(key, zero);
            }
        }
    }

    public static void main(String[] args) throws Exception
    {
        JobConf conf = new JobConf(nitproj25.class);
        conf.setJobName("nitproj25");
        conf.setOutputKeyClass(Text.class);
        conf.setOutputValueClass(IntWritable.class);
        conf.setMapperClass(Map.class);
        conf.setCombinerClass(Reduce.class);
        conf.setReducerClass(Reduce.class);
        conf.setInputFormat(TextInputFormat.class);
        conf.setOutputFormat(TextOutputFormat.class);
        FileInputFormat.setInputPaths(conf, new Path(args[0]));
        FileOutputFormat.setOutputPath(conf, new Path(args[1]));
        JobClient.runJob(conf);
    }
}
```

E. OUTPUT

```
'2121058'      0
'2131105'      0
'V101'         0
'V105'         0
'V106'         0
```

None of the doctors prescribed overpriced medicines more than or equal to 25% of the times.

VI. CONCLUSION

The focus on costs helps bring down the rate of branded drug prescriptions to below 5 percent for nearly all the drugs studied, saving the payer more than 10 percent of total pharmaceutical costs. Similar benchmark analyses are now being used in other areas, such as diagnostics, hospital contracting, and claims verification. In the dataset considered , most of the prescriptions are of generic drugs and less than 25% of times the doctors prescribe overpriced medicines , which significantly helps the hospital to bring down the cost of care.

APPENDIX

A. Setting up a single node hadoop cluster:

In the project a pseudo distributed single node hadoop cluster backed up with hadoop distributed file system has been installed and run in ubuntu linux. Hadoop is a framework written in Java for running applications on large clusters of commodity hardware and incorporates features similar to those of the Google File System (GFS) and of the MapReduce computing paradigm. Hadoop's HDFS is a highly fault tolerant distributed file system and, like Hadoop in general, designed to be deployed on low cost hardware. It provides high throughput access to application data and is suitable for applications that have large data sets. Java is needed to run hadoop on the machine. To install hadoop and run it, a dedicated hadoop system user is added, then SSH is configured and connected to the local machine. Hadoop is downloaded from Apache Download mirrors and HOME/.bashrc is updated. The environment variables are configured according to hadoop. Now hadoop installation is ready to be started. The HDFS is formatted the first time hadoop is used. [5]

B. Running the MapReduce job [6]:

Step 1: Login to the dedicated hadoop system user (hduser), start Hadoop.

```
su  hduser
startall.sh / startdfs.sh /startyarn.sh
```

Step 2: Create a .java file in your local directory using nano

```
nano project.java
```

Step 3: Create working directories in HDFS.

```
hadoop fs mkdir /user/hduser/project /user/hduser/project/input
```

Step 4: Write the input files

```
hadoop fs put ady/Documents/input.txt /user/hduser/project/input
```

Step 5: Create a temporary build directory locally

```
mkdir project_classes
```

Step 6: Compile the target Java program

```
        javac cp /home/ady/Downloads/jsoup1.8.1.jar:$HADOOP_CLASSPATH -d project_classes
/home/hduser/project.java
```

Step 7: Create a JAR archive containing the Mapper, Reducer, and driver classes

```
jar cvf project.jar C project_classes/ .
```

Step 8: Execute Hadoop

```
        hadoop jar project.jar project /user/hduser/project/input/input.txt
/user/hduser/project/output
```

REFERENCES

[1]https://www.bcgperspectives.com/content/articles/health_care_payers_providers_digital_economy_making_big_data_work_health_care_payers
[2] https://drive.google.com/file/d/0B5J-SowsoVhIS1BMRGw3WUxhqQTA/view?usp=sharing
[3] https://drive.google.com/file/d/0B5J-SowsoVhIOEdqOGNYUnlZb2c/view?usp=sharing
[4] https://www.healthkartplus.com/aboutUs
[5] http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-single-node-cluster/

[6] http://www.bogotobogo.com/Hadoop/BigData_hadoop_Install_on_ubuntu_single_node_cluster.php