

Understanding the Dataset

The dataset you provided has the following key columns relevant for a hospital readmission analysis:

encounter_id: Unique identifier for each hospital encounter.

patient_nbr: Unique identifier for each patient.

race: Patient's race.

gender: Patient's gender.

age: Patient's age group (e.g., [0-10), [10-20), etc.).

admission_type_id: Type of admission (e.g., emergency, urgent).

discharge_disposition_id: Discharge status (e.g., discharged to home, transferred to another hospital).

admission_source_id: Source of admission (e.g., referral, transfer).

time_in_hospital: Length of stay in days.

number_outpatient: Number of outpatient visits.

number_emergency: Number of emergency visits.

number_inpatient: Number of inpatient visits.

number_diagnoses: Number of diagnoses.

readmitted: Indicates if the patient was readmitted within 30 days, after 30 days, or not readmitted (NO, <30, >30).

Project Goal

The goal of this project is to perform hospital readmission analysis using Hadoop MapReduce. The output should categorize patients by age group, number of prior visits, and whether they were readmitted or not. For example:

- ✓ under30_1_0 120: 120 patients under the age of 30 with 1 prior visit were not readmitted.
- ✓ 30to60_2_1 80: 80 patients aged between 30 and 60 with 2 prior visits were readmitted.

To complete the Hospital Readmission Analysis project in Cloudera using Hadoop MapReduce, follow these detailed steps:

Steps to Perform the Analysis

1. Data Preprocessing with MapReduce

The first step involves preprocessing the data to categorize patients by age group, count their prior visits, and classify their readmission status.

Mapper: ReadmissionPreprocessingMapper.java

This mapper will extract relevant fields (age, number_outpatient, number_emergency, number_inpatient, and readmitted) from the dataset.

It will categorize patients by age group (under30, 30to60, over60), count prior visits (sum of outpatient, emergency, and inpatient visits), and classify readmission status (0 for not readmitted, 1 for readmitted).

Reducer: Not Required

This step is mainly for data transformation, so the output will be passed directly to the next step without reduction.

Driver: ReadmissionPreprocessingDriver.java

This driver will run the preprocessing job, outputting the transformed data for analysis.

2. Analysis with MapReduce

The second step is to analyze the preprocessed data and count the occurrences of each category.

Mapper: ReadmissionAnalysisMapper.java

This mapper will read the preprocessed data and emit key-value pairs where the key is a combination of age group, number of prior visits, and readmission status, and the value is 1.

Reducer: ReadmissionAnalysisReducer.java

The reducer will sum the values for each key, which gives the count of patients in each category.

Driver: ReadmissionAnalysisDriver.java

This driver will run the analysis job, producing the final output.

Prerequisites:

- 1) **Cloudera Environment:** Ensure you have Cloudera set up with Hadoop configured.
- 2) **Java Development Kit (JDK):** Make sure JDK is installed for compiling and running Java programs.
- 3) **Hadoop Installed:** Hadoop should be properly installed and configured within Cloudera.

Step 1: Set Up Your Environment

1) Access Cloudera VM:

- Open your Cloudera virtual machine and log in.
- Open the terminal (you can find it in the Applications menu or by right-clicking on the desktop and selecting "Open Terminal").

2) Create a Working Directory:

- Create a directory on your local filesystem to store your Java source files and datasets.

CODE

```
mkdir -p ~/hospital_readmission  
cd ~/hospital_readmission
```

3) Download the Dataset:

- Download the "Diabetes 130-US hospitals for years 1999-2008 Dataset" from Kaggle.
- Extract the .csv file into the hospital_readmission directory.

Step 2: Upload Dataset to HDFS

1) Create an Input Directory in HDFS:

- Create a directory in HDFS to store the input dataset.

CODE

```
hdfs dfs -mkdir -p /user/cloudera/readmission/input
```

2) Upload the Dataset to HDFS:

- Upload the .csv file to the HDFS input directory.

CODE

```
hdfs dfs -put diabetes_data.csv /user/cloudera/readmission/input
```

Step 3: Write and Compile the Java Code

1) Create Java Files:

- Create the necessary Java files for preprocessing and analysis.
- Use a text editor like gedit or vim to create the Java files.

2) Create the ReadmissionPreprocessingMapper.java:

- This mapper will preprocess the data.

CODE

gedit ReadmissionPreprocessingMapper.java

Paste the following code:

```
import java.io.IOException;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class ReadmissionPreprocessingMapper extends Mapper<LongWritable, Text, Text, Text> {

    @Override
    protected void map(LongWritable key, Text value, Context context) throws IOException,
    InterruptedException {
        String line = value.toString();
        String[] fields = line.split(",");

        // Skip the header row
        if (key.get() == 0 && line.contains("encounter_id")) {
            return;
        }

        String age = fields[4];
        try {
            int numberOutpatient = Integer.parseInt(fields[15]);
            int numberEmergency = Integer.parseInt(fields[16]);
            int numberInpatient = Integer.parseInt(fields[17]);
            String readmission = fields[49];

            int priorVisits = numberOutpatient + numberEmergency + numberInpatient;
            String ageGroup = getAgeGroup(age);
            String readmissionFlag = getReadmissionFlag(readmission);

            if (!ageGroup.isEmpty()) {
                String outputKey = ageGroup + "" + priorVisits + "" + readmissionFlag;
                context.write(new Text(outputKey), new Text("1"));
            }
        } catch (NumberFormatException e) {
            // Handle any potential number format exceptions, typically not needed if columns are
            numeric
        }
    }
}
```

```

private String getAgeGroup(String age) {
    if (age.equals("[0-10)") || age.equals("[10-20)") || age.equals("[20-30)")) {
        return "under30";
    } else if (age.equals("[30-40)") || age.equals("[40-50)") || age.equals("[50-60)")) {
        return "30to60";
    } else if (age.equals("[60-70)") || age.equals("[70-80)") || age.equals("[80-90)") ||
age.equals("[90-100)")) {
        return "over60";
    } else {
        return "";
    }
}

private String getReadmissionFlag(String readmission) {
    if (readmission.equals("NO")) {
        return "0";
    } else if (readmission.equals("<30") || readmission.equals(">30")) {
        return "1";
    } else {
        return "";
    }
}
}

```

3) Create the ReadmissionPreprocessingDriver.java:

➤ This driver will set up and run the preprocessing job.

CODE

gedit ReadmissionPreprocessingDriver.java

Paste the following code:

```

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

public class ReadmissionPreprocessingDriver {
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Readmission Preprocessing");

        job.setJarByClass(ReadmissionPreprocessingDriver.class);
        job.setMapperClass(ReadmissionPreprocessingMapper.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);
        job.setOutputFormatClass(TextOutputFormat.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}

```

4) Create the ReadmissionAnalysisMapper.java:

- This mapper will perform the analysis on the preprocessed data.

CODE

gedit ReadmissionAnalysisMapper.java

Paste the following code:

```
import java.io.IOException;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class ReadmissionAnalysisMapper extends Mapper<LongWritable, Text, Text, LongWritable> {

    private final static LongWritable one = new LongWritable(1);

    @Override
    protected void map(LongWritable key, Text value, Context context) throws IOException,
    InterruptedException {
        String line = value.toString();
        String[] parts = line.split("\t");

        if (parts.length == 2) {
            String ageVisitsReadmission = parts[0];
            context.write(new Text(ageVisitsReadmission), one);
        }
    }
}
```

5) Create the ReadmissionAnalysisReducer.java:

- This reducer will aggregate the counts for each group.

CODE

gedit ReadmissionAnalysisReducer.java

Paste the following code:

```
import java.io.IOException;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class ReadmissionAnalysisReducer extends Reducer<Text, LongWritable, Text, LongWritable> {

    @Override
    protected void reduce(Text key, Iterable<LongWritable> values, Context context)
        throws IOException, InterruptedException {
        long sum = 0;
        for (LongWritable val : values) {
            sum += val.get();
        }
        context.write(key, new LongWritable(sum));
    }
}
```

6) Create the ReadmissionAnalysisDriver.java:

- This driver will set up and run the analysis job.

CODE

gedit ReadmissionAnalysisDriver.java

Paste the following code:

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class ReadmissionAnalysisDriver {
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Readmission Analysis");

        job.setJarByClass(ReadmissionAnalysisDriver.class);
        job.setMapperClass(ReadmissionAnalysisMapper.class);
        job.setReducerClass(ReadmissionAnalysisReducer.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(LongWritable.class);
        job.setOutputFormatClass(TextOutputFormat.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

7)Compile the Java Files:

- Compile all the Java files using the following command:

CODE

```
javac -classpath $(hadoop classpath) -d . *.java
```

8)Create a JAR File:

- Package the compiled classes into a JAR file.

CODE

```
jar cf readmission_preprocessing.jar ReadmissionPreprocessing*.class
jar cf readmission_analysis.jar ReadmissionAnalysis*.class
```

Step 4: Run the Hadoop Jobs

Running the Jobs

1)Preprocess the Data:

Run `ReadmissionPreprocessingDriver.java` with the original dataset as input and an output directory for the preprocessed data.

2)Analyze the Data:

Run `ReadmissionAnalysisDriver.java` with the preprocessed data as input and an output directory for the final results.

This process will generate the desired output, where patients are categorized based on age, prior visits, and readmission status.

1) Run the Preprocessing Job:

- Execute the preprocessing job to clean and prepare the data.

CODE

```
hadoop jar readmission_preprocessing.jar ReadmissionPreprocessingDriver
/user/cloudera/readmission/input /user/cloudera/readmission/preprocessed_output
```

2) Run the Analysis Job:

- Execute the analysis job to analyze the readmission data.

CODE

```
hadoop jar readmission_analysis.jar ReadmissionAnalysisDriver
/user/cloudera/readmission/preprocessed_output /user/cloudera/readmission/analysis_output
(Single line)
```

Step 5: View and Interpret the Results

1) Check the Analysis Output:

- View the results of the analysis by checking the output files in HDFS.

CODE

```
hdfs dfs -cat /user/cloudera/readmission/analysis_output/part-r-00000
```

- The output will display the count of patients grouped by age, number of prior inpatient visits, and readmission status.

2) Interpreting the Results:

- Use the output data to identify trends and patterns related to readmission rates.
- For example, you can analyze which age groups with a specific number of prior inpatient visits have higher readmission rates.
- Analyze the output data to identify patterns related to hospital readmission rates, focusing on patient age, number of prior inpatient visits, and readmission status.

Conclusion:

By following these steps, you will have successfully completed a healthcare-related Hadoop MapReduce project in Cloudera, focusing on hospital readmission analysis for diabetic patients. This project not only helps in understanding the data processing and analysis pipeline but also provides valuable insights into healthcare data.

Updated ReadmissionPreprocessingMapper.java:

```
java Copy code

import java.io.IOException;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class ReadmissionPreprocessingMapper extends Mapper<LongWritable, Text, Text, Text> {

    @Override
    protected void map(LongWritable key, Text value, Context context) throws IOException,
        String line = value.toString();
        String[] fields = line.split(",");

        // Skip the header row
        if (key.get() == 0 && line.contains("encounter_id")) {
            return;
        }

        String age = fields[4];
        try {
            int numberOutpatient = Integer.parseInt(fields[15]);
            int numberEmergency = Integer.parseInt(fields[16]);
            int numberInpatient = Integer.parseInt(fields[17]);
            String readmission = fields[49];

            int priorVisits = numberOutpatient + numberEmergency + numberInpatient;
            String ageGroup = getAgeGroup(age);
            String readmissionFlag = getReadmissionFlag(readmission);
```

```

        if (!ageGroup.isEmpty()) {
            String outputKey = ageGroup + "_" + priorVisits + "_" + readmissionFlag;
            context.write(new Text(outputKey), new Text("1"));
        }
    } catch (NumberFormatException e) {
        // Handle any potential number format exceptions, typically not needed if colu
    }
}


private String getAgeGroup(String age) {
    if (age.equals("[0-10)") || age.equals("[10-20)") || age.equals("[20-30)") {
        return "under30";
    } else if (age.equals("[30-40)") || age.equals("[40-50)") || age.equals("[50-60)")
        return "30to60";
    } else if (age.equals("[60-70)") || age.equals("[70-80)") || age.equals("[80-90)")
        return "over60";
    } else {
        return "";
    }
}

private String getReadmissionFlag(String readmission) {
    if (readmission.equals("NO")) {
        return "0";
    } else if (readmission.equals("<30") || readmission.equals(">30")) {
        return "1";
    } else {
        return "";
    }
}
}
}

```

ReadmissionPreprocessingDriver.java

java

 Copy code

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class ReadmissionPreprocessingDriver {
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Readmission Preprocessing");

        job.setJarByClass(ReadmissionPreprocessingDriver.class);
        job.setMapperClass(ReadmissionPreprocessingMapper.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);
        job.setOutputFormatClass(TextOutputFormat.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

ReadmissionAnalysisMapper.java

```
java Copy code

import java.io.IOException;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class ReadmissionAnalysisMapper extends Mapper<LongWritable, Text, Text, LongWritable> {

    private final static LongWritable one = new LongWritable(1);

    @Override
    protected void map(LongWritable key, Text value, Context context) throws IOException,
        String line = value.toString();
        String[] parts = line.split("\\t");

        if (parts.length == 2) {
            String ageVisitsReadmission = parts[0];
            context.write(new Text(ageVisitsReadmission), one);
        }
    }
}
```

ReadmissionAnalysisReducer.java

```
java Copy code


import java.io.IOException;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class ReadmissionAnalysisReducer extends Reducer<Text, LongWritable, Text, LongWritable> {

    @Override
    protected void reduce(Text key, Iterable<LongWritable> values, Context context)
        throws IOException, InterruptedException {
        long sum = 0;
        for (LongWritable val : values) {
            sum += val.get();
        }
        context.write(key, new LongWritable(sum));
    }
}
```


ReadmissionAnalysisDriver.java

java

 Copy code

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class ReadmissionAnalysisDriver {
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Readmission Analysis");

        job.setJarByClass(ReadmissionAnalysisDriver.class);
        job.setMapperClass(ReadmissionAnalysisMapper.class);
        job.setReducerClass(ReadmissionAnalysisReducer.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(LongWritable.class);
        job.setOutputFormatClass(TextOutputFormat.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

