# Metropolitan State University

ICS 613: Introduction to Big Data Computing Systems

Fall 2018

Analysis of Google Play Store Apps

Michael Ayako and Abidemi Lawal

# Introduction

In this project, we are going to analyze a data set on google play store apps targeting various questions for analysis purposes. We will answer the following questions:

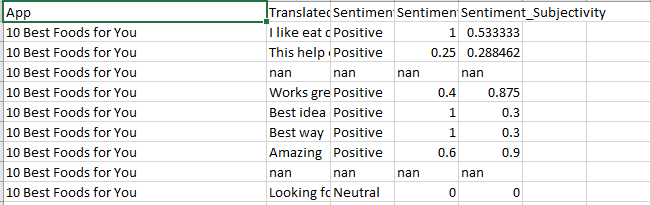
* Does the type of App affect the Sentiment of the users? Are the customers using free apps more satisfied than the customers using paid apps?
* Does the genre of the App affect the sentiments of the user?
* What are the Top 10 genres and Bottom 10 Genres regarding how many apps fall under each genre?
* What are the top 10 categories and the bottom 10 categories in terms of ratings?

# Data set description

The data set we are using consists of the following files:

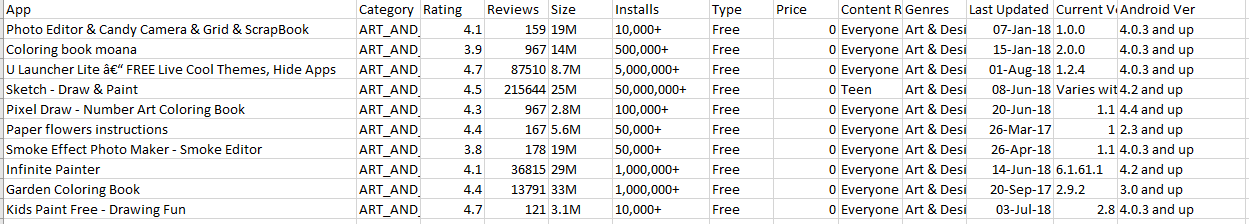
* File 1: This file consists of 100 ratings per every application in the play store. It has various fields that would be used to perform join and pull out data of user sentiments that would be valuable for analysis. Fields in the data file include:

**App|Translated\_Review|Sentiment|Sentiment\_Polarity|Sentiment\_Subjectivity**



* File 2: In this file data on the various applications details are stored. This file consists of detailed application information of the apps on Google Play Store. Fields include:

**App| Category| Rating| Reviews| Size| Installs| Type| Price| Content Rating| Genre| Last Updated| Android Version**



### Data preparation:

File 1 consists of 5 columns, however in our analysis we are going to use only the following attributes.

|  |  |
| --- | --- |
| App | This is the name of the applications that are in the google plays store (will mostly be used as our key in any join we perform throughout the analysis) |
| Sentiment | This represents how the users felt regarding a certain application. I |

File 2 consists of 12 columns, but we shall be focusing on the following attributes:

|  |  |
| --- | --- |
| App | This is the name of the applications that are in the google plays store (will mostly be used as our key in any join we perform throughout the analysis) |
| Category | This field represents the application category. It states what kind of application has been placed on the app store e.g. Art & Design, Communication etc. |
| Rating | This is the application rating as it stands on google play at the time the data was collected from the google play store |
| Type | This contains data on whether an application is free or is a sold application. |
| Genre | This represent what kinds of genre the applications are designed to be. |

### Data cleaning:

These are the steps/code that we used to extract our fields of interest from the original input files.

Our data is in the .csv format, so we basically used the Microsoft Excel to clean our data so that we can work with it in the course of our analysis. We performed the following actions

* We deleted the unnecessary columns that we in our data sets
* We deleted the first row in both datasets that had the titles of the columns

# Analysis

## Analysis task 1: Does the type of App affect the Sentiment of the users? Are the customers using free apps more satisfied than the customers using paid apps?

The goal of this task is to connect an Application – Type to an Application-Sentiment. The Key that would be used to link this up would be the Application Name. The end goal is to output positive, negative and neutral sentiment count for each “type” and use it to plot our result.

We will implement this task using two Hadoop map-reduce jobs as follows:

### Job 1:

|  |  |
| --- | --- |
| **Map input:** | ***File A: linenum,(App, sentiment)***  ***File B: linenum, (App, type)*** |
| **Mapper filter:** | ***No filter shall be used*** |
| **Map output:** | ***App, (type, sentiment)*** |
| **Reduce input:** | ***App, (type, sentiment)*** |
| **Reduce function:** | ***We wanted each type to correlate with its sentiment value. We shall only omit the Key “App” and Pass out “type” as our new key and “sentiment” as our value*** |
| **Reduce output:** | ***Type,(Sentiment)*** |

### Job 2:

|  |  |
| --- | --- |
| **Map input:** | ***Type,(Sentiment)*** |
| **Mapper filter:** | ***No action to be performed*** |
| **Map output:** | ***Type,(Sentiment)*** |
| **Reduce input:** | ***Type,(Sentiment)*** |
| **Reduce function:** | ***Count the sentiment values and output sentiment count in terms of positive and negative*** |
| **Reduce output:** | ***Type,(Sentiment count)*** |

### Findings:

The main findings of this chart indicate that regarding the data we used for analysis both kinds of customers that access the 2 types of application are content with them. It’s also observable that most users are lean towards the neutral side when it comes to the paid version of the application.

## Analysis task 2: Does the genre of the App affect the sentiments of the user?

The aim of this computation is show how genres differ in terms of positive to negative sentiments. It will be used to plot the final graph in order to show the various outputs.

We will implement this task using two Hadoop map-reduce jobs as follows:

### Job 1:

|  |  |
| --- | --- |
| **Map input:** | ***File A: linenum,(App, sentiment)***  ***File B: linenum, (App, genre)*** |
| **Mapper filter:** | ***No filter shall be used*** |
| **Map output:** | ***App, (genre, sentiment)*** |
| **Reduce input:** | ***App, (genre, sentiment)*** |
| **Reduce function:** | ***We wanted each type to correlate with its sentiment value. We shall only omit the Key “App” and Pass out “genre” as our new key and “sentiment” as our value*** |
| **Reduce output:** | ***Type,(Sentiment)*** |

### Job 2:

|  |  |
| --- | --- |
| **Map input:** | ***Genre, (Sentiment)*** |
| **Mapper filter:** | ***No action to be performed*** |
| **Map output:** | ***Genre, (Sentiment)*** |
| **Reduce input:** | ***Genre, (Sentiment)*** |
| **Reduce function:** | ***Count the sentiment values and output sentiment count in terms of positive and negative*** |
| **Reduce output:** | ***Genre, (Sentiment count)*** |

### Findings:

## Analysis task 3: Finding the Top 10 genres and Bottom 10 Genres regarding how many apps fall under each genre

The goal of this task is to find which genre category do most App developers aim for when they develop their application. This will also show which genre receive less attention in the development process.

This will be implemented using PySpark

### Detailed explanation of steps:

* Using the 2nd File, we will use excel to delete all irrelevant columns and remain with only 2 columns for this analysis i.e. “App” and “Genre” and split
* We mapped the Genre as the Key and the app we replaced it with a value of 1 so that we would be able to count it later
* We then used **reduceByKey()** to add up all the apps under each genre
* Finally, we used **sortBy()** to arrange in both ascending and descending order and used the **take(10)** to get only ten values from each sort performed

### Findings:

## Analysis task 4: Find the top 10 categories and the bottom 10 categories in terms of ratings

The aim of this analysis task is to find the average rating of each category and then highlight the top and bottom ten categories.

We shall implement this using PySpark

### Detailed explanation of steps:

* File 2 shall be used for the analysis. We went ahead and edited the file in excel so we are only left with the “category” and “rating” column.
* We mapped a key/value pair: category as the key and 2 values namely:

1. Rating
2. 1 – this will be used to count the number of apps in a category

* We then used **reduceByKey()** to sum up the values or the ratings and count of apps in each category
* We then used **mapValues()** to get the average of each category by dividing each rating by the number of apps in that category
* Finally, we used **sortBy()** to arrange in ascending and descending order then **take(10)** to get values from each side respectively.

### Findings:

During the implementation we noticed an error in record for this data set where by there was a category that displayed (1.9) as category name and (19) as the rating so we disregarded this record and selected the 10 underneath it.

# APPENDICES

## Analysis task 1:

### Workflow.xml

<workflow-app xmlns="uri:oozie:workflow:0.1" name="Free\_or\_Paid">

<start to="Type\_Sentiment\_Join-node"/>

<action name="Type\_Sentiment\_Join-node">

<map-reduce>

<job-tracker>${resourceManager}</job-tracker>

<name-node>${nameNode}</name-node>

<prepare>

<delete path="${outputDir1}"/>

</prepare>

<configuration>

<property>

<name>mapred.mapper.new-api</name>

<value>true</value>

</property>

<property>

<name>mapred.reducer.new-api</name>

<value>true</value>

</property>

<property>

<name>mapreduce.job.map.class</name>

<value>mapper1</value>

</property>

<property>

<name>mapreduce.job.output.key.class</name>

<value>org.apache.hadoop.io.Text</value>

</property>

<property>

<name>mapreduce.job.output.value.class</name>

<value>compkey1</value>

</property>

<property>

<name>mapreduce.job.reduce.class</name>

<value>reducer1</value>

</property>

<property>

<name>mapreduce.input.fileinputformat.inputdir</name>

<value>${inputDir}</value>

</property>

<property>

<name>mapreduce.output.fileoutputformat.outputdir</name>

<value>${outputDir1}</value>

</property>

</configuration>

</map-reduce>

<ok to="\_Count-node"/>

<error to="fail"/>

</action>

<action name="\_Count-node">

<map-reduce>

<job-tracker>${resourceManager}</job-tracker>

<name-node>${nameNode}</name-node>

<prepare>

<delete path="${outputDir2}"/>

</prepare>

<configuration>

<property>

<name>mapred.mapper.new-api</name>

<value>true</value>

</property>

<property>

<name>mapred.reducer.new-api</name>

<value>true</value>

</property>

<property>

<name>mapreduce.job.map.class</name>

<value>mapper2</value>

</property>

<property>

<name>mapreduce.job.output.key.class</name>

<value>org.apache.hadoop.io.Text</value>

</property>

<property>

<name>mapreduce.job.output.value.class</name>

<value>org.apache.hadoop.io.Text</value>

</property>

<property>

<name>mapreduce.job.reduce.class</name>

<value>reducer2</value>

</property>

<property>

<name>mapreduce.input.fileinputformat.inputdir</name>

<value>${outputDir1}</value>

</property>

<property>

<name>mapreduce.output.fileoutputformat.outputdir</name>

<value>${outputDir2}</value>

</property>

</configuration>

</map-reduce>

<ok to="end"/>

<error to="fail"/>

</action>

<kill name="fail">

<message>Map/Reduce failed, error message[${wf:errorMessage(wf:lastErrorNode())}]</message>

</kill>

<end name="end"/>

</workflow-app>

### Job.properties

nameNode=hdfs://localhost:8020

resourceManager=localhost:8032

queueName=default

baseDir=${nameNode}/user/cloudera/Proj1

oozie.wf.application.path=${baseDir}/job

oozie.libpath=${oozie.wf.application.path}/lib

inputDir=${baseDir}/input

outputDir1=${baseDir}/output1

outputDir2=${baseDir}/output2

### driver1.java

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;

public class driver1 {

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

Job job = new Job();

job.setJarByClass(driver1.class);

job.setJobName("Job 1");

//to accept the hdfs input and output dir at run time

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

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

//setting the class names

job.setMapperClass(mapper1.class);

job.setReducerClass(reducer1.class);

//setting the output data type classes

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(compkey1.class);

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

}

}

### Driver2.java

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;

public class driver2 {

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

Job job = new Job();

job.setJarByClass(driver2.class);

job.setJobName("Positive");

//to accept the hdfs input and output dir at run time

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

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

//setting the class names

job.setMapperClass(mapper2.class);

job.setReducerClass(reducer2.class);

//setting the output data type classes

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(Text.class);

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

}

}

### compkey1.java

**import** java.io.DataInput;

**import** java.io.DataOutput;

**import** java.io.IOException;

**import** org.apache.hadoop.io.WritableComparable;

**public** **class** compkey1 **implements** WritableComparable<compkey1>{

String type,sentiment;

**int** fileid=0;

**public** compkey1(){

}

**public** compkey1(String type1,String sentiment1,**int** fileid1){

type = type1;

sentiment = sentiment1;

fileid = fileid1;

}

@Override

**public** **void** readFields(DataInput in)**throws** IOException{

type = in.readUTF();

sentiment = in.readUTF();

fileid = in.readInt();

}

@Override

**public** **void** write(DataOutput out)**throws** IOException{

out.writeUTF(type);

out.writeUTF(sentiment);

out.writeInt(fileid);

}

**public** **int** compareTo(compkey1 o){

// **TODO** Auto-generated method stub

**int** q = 0;

**if**(type.contentEquals(o.type)) q = -1;

**else**{

**if**(sentiment.contentEquals(o.sentiment)) q = -1;

**else**{

**if**(fileid > o.fileid) q = -1;

**else** **if**(fileid < o.fileid) q = -1;

**else** q = 0;

}

}

**return** q;

}

**public** String toString(){

**return** "" + fileid + " " + type + " " + sentiment;

}

@Override

**public** **int** hashCode() {

**final** **int** prime = 31;

**int** result = 1;

result = prime \* result + fileid;

result = prime \* result

+ ((sentiment == **null**) ? 0 : sentiment.hashCode());

result = prime \* result + ((type == **null**) ? 0 : type.hashCode());

**return** result;

}

@Override

**public** **boolean** equals(Object obj) {

**if** (**this** == obj)

**return** **true**;

**if** (obj == **null**)

**return** **false**;

**if** (getClass() != obj.getClass())

**return** **false**;

compkey1 other = (compkey1) obj;

**if** (fileid != other.fileid)

**return** **false**;

**if** (sentiment == **null**) {

**if** (other.sentiment != **null**)

**return** **false**;

} **else** **if** (!sentiment.equals(other.sentiment))

**return** **false**;

**if** (type == **null**) {

**if** (other.type != **null**)

**return** **false**;

} **else** **if** (!type.equals(other.type))

**return** **false**;

**return** **true**;

}

}

### Compkey2.java

import java.io.DataInput;

import java.io.DataOutput;

import java.io.IOException;

import org.apache.hadoop.io.WritableComparable;

public class compkey2 implements WritableComparable<compkey2>{

int pos,neg,nan;

public compkey2(){}

public compkey2(int pos1,int neg1,int nan1){

pos = pos1;

neg = neg1;

nan = nan1;

}

@Override

public void readFields(DataInput in) throws IOException {

// TODO Auto-generated method stub

pos = in.readInt();

neg = in.readInt();

nan = in.readInt();

}

@Override

public void write(DataOutput out) throws IOException {

// TODO Auto-generated method stub

out.writeInt(pos);

out.writeInt(neg);

out.writeInt(nan);

}

@Override

public int compareTo(compkey2 o) {

// TODO Auto-generated method stub

int q;

if(pos > o.pos) q = -1;

else if(pos < o.pos) q = -1;

else{

if(neg > o.neg) q = -1;

else if(neg < o.neg) q = -1;

else{

if(nan > o.nan) q = -1;

else if(nan < o.nan) q = -1;

else q = 0;

}

}

return q;

}

public String toString(){

return "Positive Reviews = " + pos + ", Negative Reviews " + neg + ", Not reviewed or Nuetral = " + nan;

}

@Override

public int hashCode() {

final int prime = 31;

int result = 1;

result = prime \* result + nan;

result = prime \* result + neg;

result = prime \* result + pos;

return result;

}

@Override

public boolean equals(Object obj) {

if (this == obj)

return true;

if (obj == null)

return false;

if (getClass() != obj.getClass())

return false;

compkey2 other = (compkey2) obj;

if (nan != other.nan)

return false;

if (neg != other.neg)

return false;

if (pos != other.pos)

return false;

return true;

}

}

### Mapper1.java

import java.io.IOException;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class mapper1 extends Mapper <LongWritable,Text, Text,compkey1> {

public void map(LongWritable key, Text value, Context context) throws IOException,InterruptedException{

String app = "",type= "",sentiment="";

String[] v = value.toString().split(",");

String[] q = value.toString().split(",");

int fileid= -1;

if(v.length == 2){

app = v[0];

sentiment = v[1];

fileid=1;

}

if(q.length == 4){

app = q[0];

type =q[3];

fileid=2;

}

if( fileid > 0 ) {

context.write(new Text(app), new compkey1(type,sentiment,fileid));

}

}

}

### Reducer1.java

import java.io.IOException;

import java.util.ArrayList;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

public class reducer1 extends Reducer<Text,compkey1,Text,Text> {

public void reduce(Text key, Iterable<compkey1> values, Context context)

throws IOException, InterruptedException {

ArrayList<String> sentiments = new ArrayList<String>();

int count =0;

String type = "";

for(compkey1 v : values){

if(v.fileid == 2){

type = v.type;

break;

}

}

for(compkey1 v : values){

if(v.fileid == 1){

sentiments.add(v.sentiment);

count++;

}

}

for(int i=0;i<count;i++){

context.write(new Text(type), new Text(sentiments.get(i)));

}

}

}

### Mapper2.java

import java.io.IOException;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class mapper2 extends Mapper <LongWritable, Text, Text,Text>{

public void map(LongWritable key, Text value, Context context) throws IOException,InterruptedException{

String[] v = value.toString().split("\\t");

context.write(new Text(v[0]), new Text(v[1]));

}

}

### Reducer2.java

import java.io.IOException;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

public class reducer2 extends Reducer <Text,Text,Text,compkey2> {

public void reduce(Text key, Iterable<Text> values, Context context)

throws IOException, InterruptedException {

int poscount=0,negcount=0,nancount=0;

for(Text val : values){

if(val.toString().contentEquals("Positive"))

poscount++;

if(val.toString().contentEquals("Negative"))

negcount++;

if(val.toString().contentEquals("nan") || val.toString().contentEquals("Nuetral"))

nancount++;

}

if(nancount>0)

context.write(new Text(key), new compkey2(poscount,negcount,nancount));

}

}

## Analysis Task 2:

### Workflow.xml

<workflow-app xmlns="uri:oozie:workflow:0.1" name="Free\_or\_Paid">

<start to="Genre\_Sentiment\_Join-node"/>

<action name="Genre\_Sentiment\_Join-node">

<map-reduce>

<job-tracker>${resourceManager}</job-tracker>

<name-node>${nameNode}</name-node>

<prepare>

<delete path="${outputDir1}"/>

</prepare>

<configuration>

<property>

<name>mapred.mapper.new-api</name>

<value>true</value>

</property>

<property>

<name>mapred.reducer.new-api</name>

<value>true</value>

</property>

<property>

<name>mapreduce.job.map.class</name>

<value>mapper1</value>

</property>

<property>

<name>mapreduce.job.output.key.class</name>

<value>org.apache.hadoop.io.Text</value>

</property>

<property>

<name>mapreduce.job.output.value.class</name>

<value>compkey1</value>

</property>

<property>

<name>mapreduce.job.reduce.class</name>

<value>reducer1</value>

</property>

<property>

<name>mapreduce.input.fileinputformat.inputdir</name>

<value>${inputDir}</value>

</property>

<property>

<name>mapreduce.output.fileoutputformat.outputdir</name>

<value>${outputDir1}</value>

</property>

</configuration>

</map-reduce>

<ok to="\_Count-node"/>

<error to="fail"/>

</action>

<action name="\_Count-node">

<map-reduce>

<job-tracker>${resourceManager}</job-tracker>

<name-node>${nameNode}</name-node>

<prepare>

<delete path="${outputDir2}"/>

</prepare>

<configuration>

<property>

<name>mapred.mapper.new-api</name>

<value>true</value>

</property>

<property>

<name>mapred.reducer.new-api</name>

<value>true</value>

</property>

<property>

<name>mapreduce.job.map.class</name>

<value>mapper2</value>

</property>

<property>

<name>mapreduce.job.output.key.class</name>

<value>org.apache.hadoop.io.Text</value>

</property>

<property>

<name>mapreduce.job.output.value.class</name>

<value>org.apache.hadoop.io.Text</value>

</property>

<property>

<name>mapreduce.job.reduce.class</name>

<value>reducer2</value>

</property>

<property>

<name>mapreduce.input.fileinputformat.inputdir</name>

<value>${outputDir1}</value>

</property>

<property>

<name>mapreduce.output.fileoutputformat.outputdir</name>

<value>${outputDir2}</value>

</property>

</configuration>

</map-reduce>

<ok to="end"/>

<error to="fail"/>

</action>

<kill name="fail">

<message>Map/Reduce failed, error message[${wf:errorMessage(wf:lastErrorNode())}]</message>

</kill>

<end name="end"/>

</workflow-app>

### Job.properties

NameNode=hdfs://localhost:8020

resourceManager=localhost:8032

queueName=default

baseDir=${nameNode}/user/cloudera/Proj2

oozie.wf.application.path=${baseDir}/job

oozie.libpath=${oozie.wf.application.path}/lib

inputDir=${baseDir}/input

outputDir1=${baseDir}/output1

outputDir2=${baseDir}/output2

#### Driver1.java

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;

public class driver1 {

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

Job job = new Job();

job.setJarByClass(driver1.class);

job.setJobName("Job 1");

//to accept the hdfs input and output dir at run time

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

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

//setting the class names

job.setMapperClass(mapper1.class);

job.setReducerClass(reducer1.class);

//setting the output data type classes

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(compkey1.class);

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

}

}

### Driver2.java

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;

public class driver2 {

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

Job job = new Job();

job.setJarByClass(driver2.class);

job.setJobName("Count Job");

//to accept the hdfs input and output dir at run time

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

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

//setting the class names

job.setMapperClass(mapper2.class);

job.setReducerClass(reducer2.class);

//setting the output data type classes

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(Text.class);

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

}

}

### Compkey1.java

import java.io.DataInput;

import java.io.DataOutput;

import java.io.IOException;

import org.apache.hadoop.io.WritableComparable;

public class compkey1 implements WritableComparable<compkey1>{

String genre,sentiment;

int fileid=0;

public compkey1(){

}

public compkey1(String genre1,String sentiment1,int fileid1){

genre = genre1;

sentiment = sentiment1;

fileid = fileid1;

}

@Override

public void readFields(DataInput in)throws IOException{

genre = in.readUTF();

sentiment = in.readUTF();

fileid = in.readInt();

}

@Override

public void write(DataOutput out)throws IOException{

out.writeUTF(genre);

out.writeUTF(sentiment);

out.writeInt(fileid);

}

public int compareTo(compkey1 o){

// TODO Auto-generated method stub

int q = 0;

if(genre.contentEquals(o.genre)) q = -1;

else{

if(sentiment.contentEquals(o.sentiment)) q = -1;

else{

if(fileid > o.fileid) q = -1;

else if(fileid < o.fileid) q = -1;

else q = 0;

}

}

return q;

}

public String toString(){

return "" + fileid + " " + genre + " " + sentiment;

}

@Override

public int hashCode() {

final int prime = 31;

int result = 1;

result = prime \* result + fileid;

result = prime \* result

+ ((sentiment == null) ? 0 : sentiment.hashCode());

result = prime \* result + ((genre == null) ? 0 : genre.hashCode());

return result;

}

@Override

public boolean equals(Object obj) {

if (this == obj)

return true;

if (obj == null)

return false;

if (getClass() != obj.getClass())

return false;

compkey1 other = (compkey1) obj;

if (fileid != other.fileid)

return false;

if (sentiment == null) {

if (other.sentiment != null)

return false;

} else if (!sentiment.equals(other.sentiment))

return false;

if (genre == null) {

if (other.genre != null)

return false;

} else if (!genre.equals(other.genre))

return false;

return true;

}

}

### Compkey2.java

import java.io.DataInput;

import java.io.DataOutput;

import java.io.IOException;

import org.apache.hadoop.io.WritableComparable;

public class compkey2 implements WritableComparable<compkey2>{

int pos,neg,nan;

public compkey2(){}

public compkey2(int pos1,int neg1,int nan1){

pos = pos1;

neg = neg1;

nan = nan1;

}

@Override

public void readFields(DataInput in) throws IOException {

// TODO Auto-generated method stub

pos = in.readInt();

neg = in.readInt();

nan = in.readInt();

}

@Override

public void write(DataOutput out) throws IOException {

// TODO Auto-generated method stub

out.writeInt(pos);

out.writeInt(neg);

out.writeInt(nan);

}

@Override

public int compareTo(compkey2 o) {

// TODO Auto-generated method stub

int q;

if(pos > o.pos) q = -1;

else if(pos < o.pos) q = -1;

else{

if(neg > o.neg) q = -1;

else if(neg < o.neg) q = -1;

else{

if(nan > o.nan) q = -1;

else if(nan < o.nan) q = -1;

else q = 0;

}

}

return q;

}

public String toString(){

return "Positive Reviews = " + pos + ", Negative Reviews " + neg + ", Not reviewed or Nuetral = " + nan;

}

@Override

public int hashCode() {

final int prime = 31;

int result = 1;

result = prime \* result + nan;

result = prime \* result + neg;

result = prime \* result + pos;

return result;

}

@Override

public boolean equals(Object obj) {

if (this == obj)

return true;

if (obj == null)

return false;

if (getClass() != obj.getClass())

return false;

compkey2 other = (compkey2) obj;

if (nan != other.nan)

return false;

if (neg != other.neg)

return false;

if (pos != other.pos)

return false;

return true;

}

}

### Mapper1.java

import java.io.IOException;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class mapper1 extends Mapper <LongWritable,Text, Text,compkey1> {

public void map(LongWritable key, Text value, Context context) throws IOException,InterruptedException{

String app = "",genre= "",sentiment="";

String[] v = value.toString().split(",");

String[] q = value.toString().split(",");

int fileid= -1;

if(v.length == 2){

app = v[0];

sentiment = v[1];

fileid=1;

}

if(q.length == 5){

app = q[0];

genre =q[1];

fileid=2;

}

if( fileid > 0 ) {

context.write(new Text(app), new compkey1(genre,sentiment,fileid));

}

}

}

### Reducer1.java

import java.io.IOException;

import java.util.ArrayList;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

public class reducer1 extends Reducer<Text,compkey1,Text,Text> {

public void reduce(Text key, Iterable<compkey1> values, Context context)

throws IOException, InterruptedException {

ArrayList<String> sentiments = new ArrayList<String>();

int count =0;

String genre = "";

for(compkey1 v : values){

if(v.fileid == 2){

genre = v.genre;

break;

}

}

for(compkey1 v : values){

if(v.fileid == 1){

sentiments.add(v.sentiment);

count++;

}

}

for(int i=0;i<count;i++){

context.write(new Text(genre), new Text(sentiments.get(i)));

}

}

}

### Mapper2.java

import java.io.IOException;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class mapper2 extends Mapper <LongWritable, Text, Text,Text>{

public void map(LongWritable key, Text value, Context context) throws IOException,InterruptedException{

String[] v = value.toString().split("\\t");

context.write(new Text(v[0]), new Text(v[1]));

}

}

### Reducer2.java

import java.io.IOException;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

public class reducer2 extends Reducer <Text,Text,Text,compkey2> {

public void reduce(Text key, Iterable<Text> values, Context context)

throws IOException, InterruptedException {

int poscount=0,negcount=0,nancount=0;

for(Text val : values){

if(val.toString().contentEquals("Positive"))

poscount++;

if(val.toString().contentEquals("Negative"))

negcount++;

if(val.toString().contentEquals("nan") || val.toString().contentEquals("Nuetral"))

nancount++;

}

if(nancount>0)

context.write(new Text(key), new compkey2(poscount,negcount,nancount));

}

}

## Analysis Task 3:

PlaystoreRdd = sc.textFile("file:/home/cloudera/Proj3/googleplaystore\_edited3.csv")

playstorerddsplit = PlaystoreRdd.map(lambda x : x.split(","))

GenreKey = playstorerddsplit.map(lambda x : (x[1],1))

GenreKeyCount = GenreKey.reduceByKey(lambda x,y : x+y)

for i in GenreKeyCount.sortBy(lambda (x,y) : -y).take(10):print i

...

for i in GenreCount.sortBy(lambda (x,y) : y).take(10):print i

...

## Analysis Task 4:

PlaystoreRdd = sc.textFile("file:/home/cloudera/Proj4/googleplaystore\_edited4.csv")

playstorerddsplit = PlaystoreRdd.map(lambda x : x.split(","))

RatingKey = playstorerddsplit.map(lambda x : (x[0],(x[1],1)))

RatingKeyReduce = RatingKey.reduceByKey(lambda (x1,y1),(x2,y2) : ((float)(x1)+(float)(x2),(int)(y1)+(int)(y2)))

RatingsAverage = RatingKeyReduce.mapValues(lambda (x,y) : (float)(x) / y)

for i in RatingsAverage.sortBy(lambda (x,y) : -y).take(10):print i

...

for i in RatingsAverage.sortBy(lambda (x,y) : y).take(10):print i

...