**Project Report**

**Problem Statement:**

The dataset on which I worked on is the “Yelp Dataset”. The dataset has thousands of business attributes which can be helpful in broadening the analysis and implement multiple design patterns. The dataset is in JSON format, so I first converted the JSON files to CSV files using a python script available on <https://www.yelp.com/dataset_challenge>.

Looking at the dataset few analysis question that I have performed are as below:

1. Simple Random Sampling – As the dataset is very large so I performed to SRS to get some random data to perform analysis.
2. Partitioning (Data Organizational Pattern) – To narrow the analysis to particular year I partitioned the dataset by year so as to perform analysis only for particular year.
3. Summarization Pattern – Each user has given single or multiple reviews to particular business ID’s. So I calculated the total count of reviews given by each user and his first and last review date.
4. Reduce side Inner Join (Join Pattern) – Joined two datasets on business ID’s and then found the top 25 cities with most user reviews.
5. Implemented Content based recommendation algorithm to find correlation between the pair of business ID’s using cosine similarity to find the similarity score for the pair and then sorting score in descending order.
6. Implemented Sentiments analysis and average rating received by business ID’s in particular month. From the analysis we can predict that in which month the sentiment reviews and rating were good or bad which can help to understand behavior of user visiting restaurants in particular period.

**Link to Dataset:** <https://www.yelp.com/dataset_challenge>.

**Description of Analysis:**

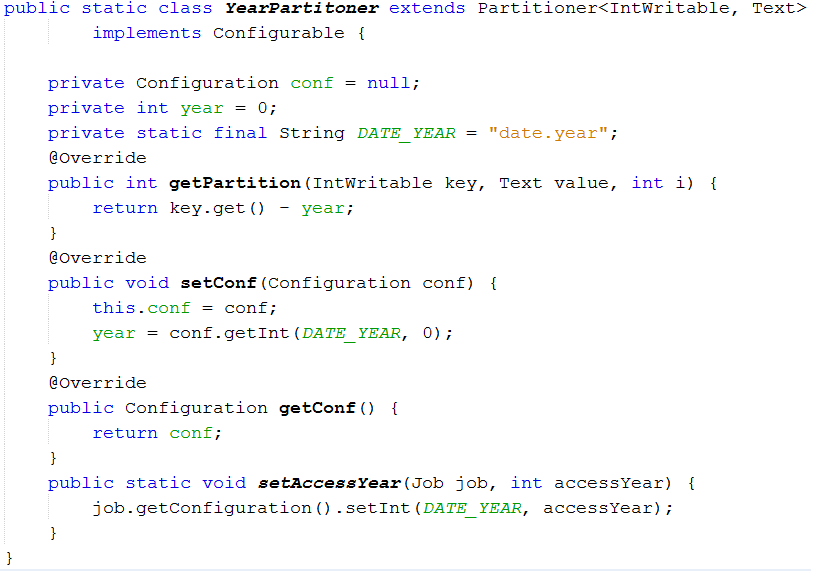
1. ***Simple Random Sampling*** – As the dataset is very large so I performed to SRS to get some random data as the probability of data getting selected is same to perform analysis.
2. ***Partitioning (Data Organizational Pattern)*** – To narrow the analysis to particular year I partitioned the dataset by year. The file on which I implemented this pattern had entry’s starting from year 2009 to current 2016 so there will be 8 partitions one for each year. The number of reduce task had to set in the driver function before running the job. In the Mapper function key will the year which is fetched by parsing the file and the value emitted from mapper will entire value coming as input to mapper. We will need Partitioner for the job as it will determine which reducer to send the partitioned data. This partitioned data can be further used for the analysis.

**Code Snippets:**

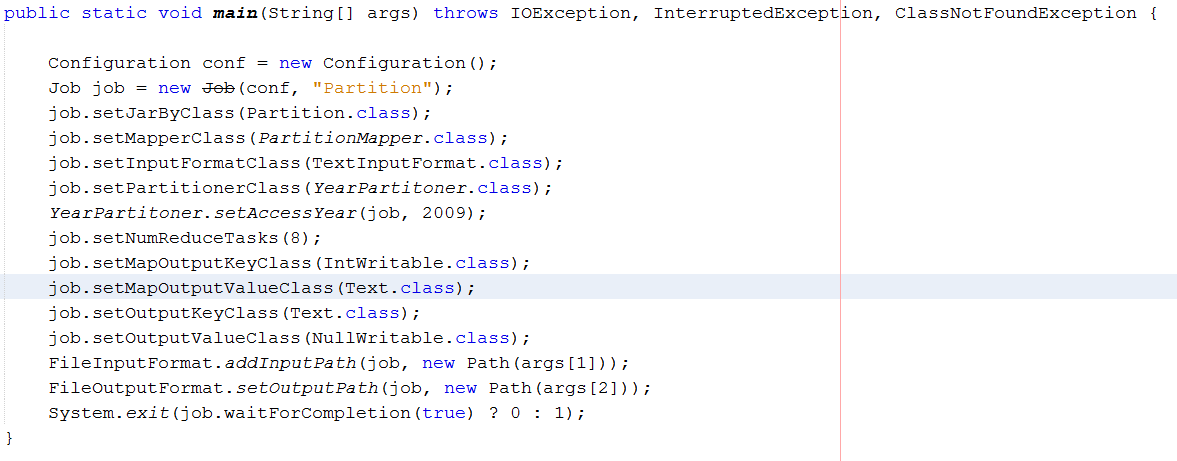
**Mapper function**

****

**Partitioner function**



**Driver function**

****

1. **Recommendation Algorithm:** Implemented the content based recommendation algorithm to find correlation between the business ID pairs and count the number of users who rated both the business ID’s.

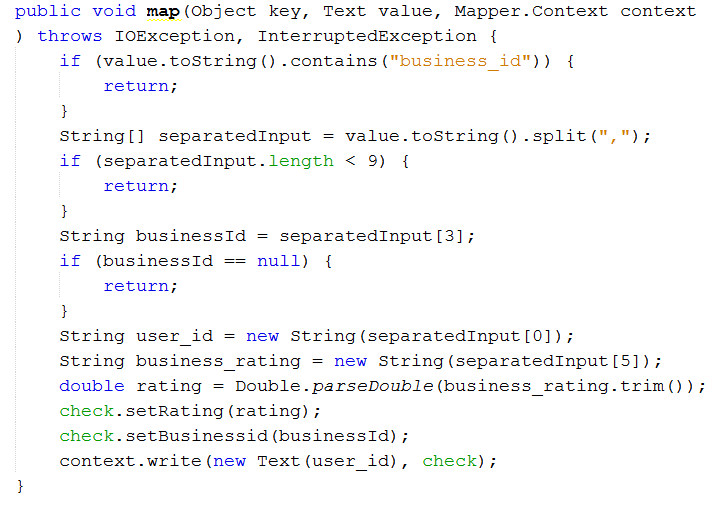
**Step1:**

Mapper:

Extract the user id, business id and rating given by user.

Input -> data file

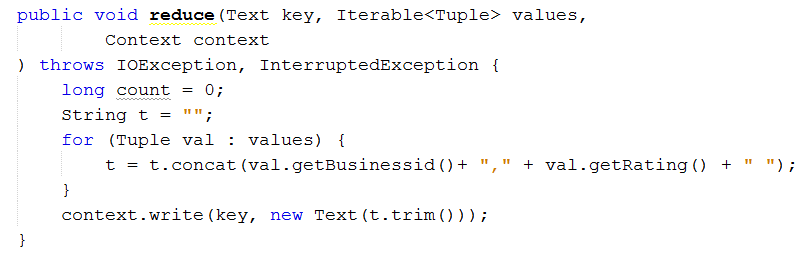
Output -> userID ,(businessID,rating)



Reducer: Group userID,(businessID,rating)

Input -> userID,(businessID,rating)

Output-> userID, [(businessID,rating)( businessID,rating)…..]



**Final output sample of Step 1**:

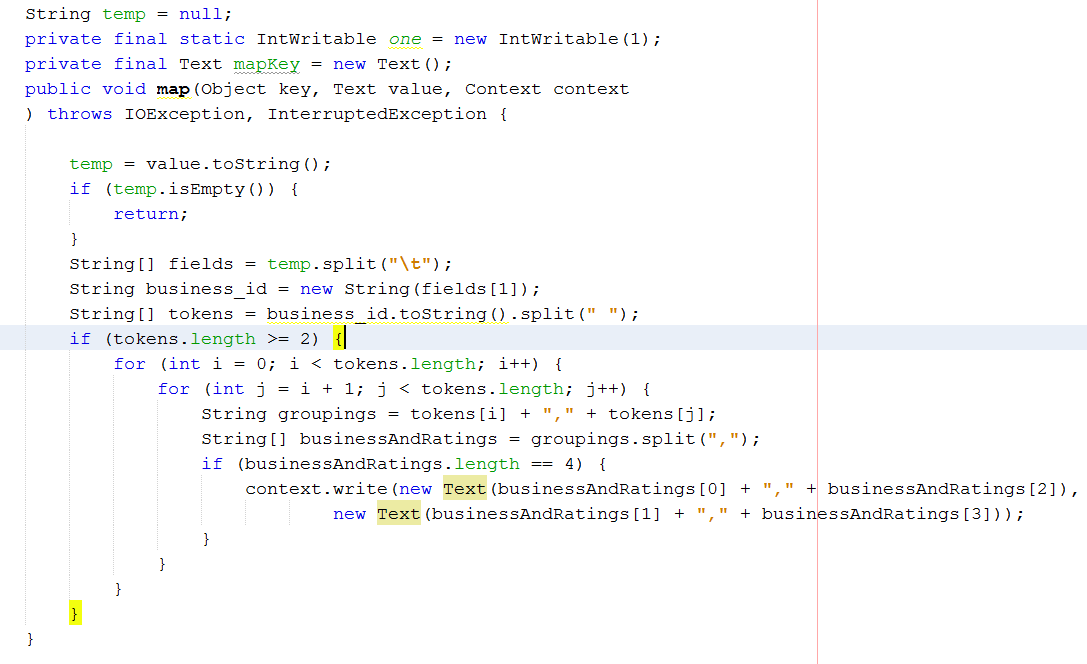
00pwvsiP3pYRwHtmuR9x8Q QMX1ssofSveU3qhVAKLnXA,3.0 utyqO\_IVBAN8ULZmi52XaA,5.0 utyqO\_IVBAN8ULZmi52XaA,3.0

**Step2:**

Mapper: Create business, rating pairs for each user (combinations)

Input – > userID, [(businessID,ratings) (businessID,ratings)…]

Output -> (businessID, businessID) , (rating, rating)

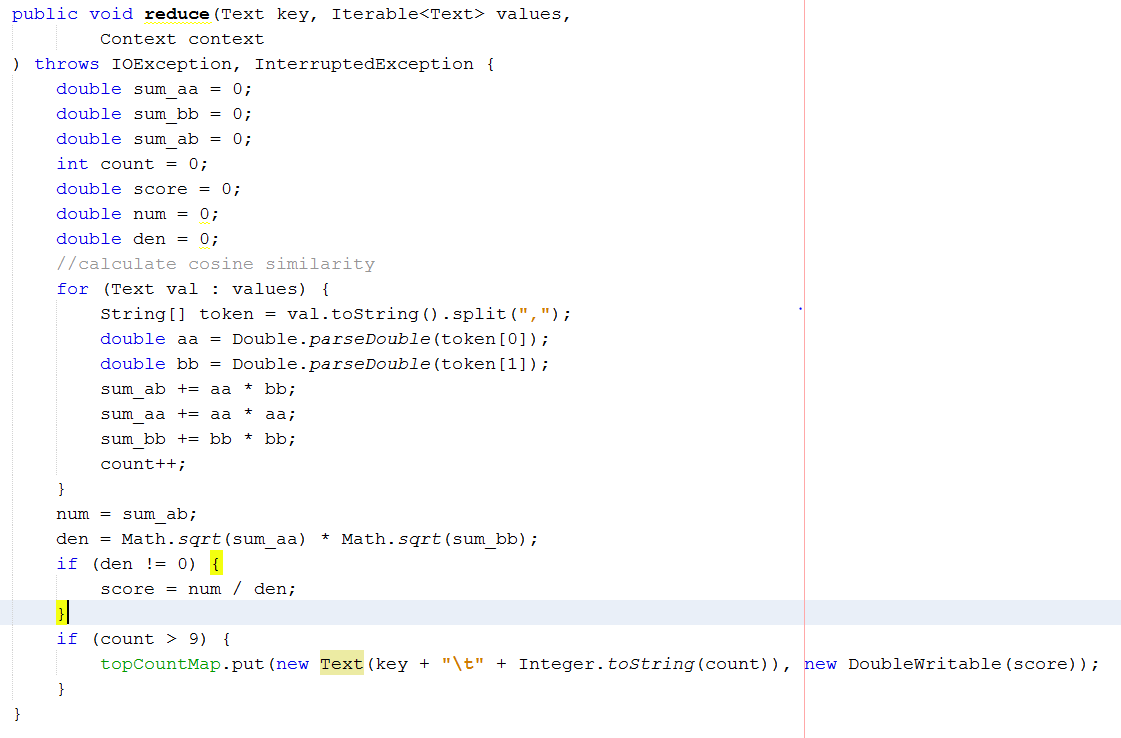


Reducer:

Compute rating based similarity for each business pair (Cosine Similarity)

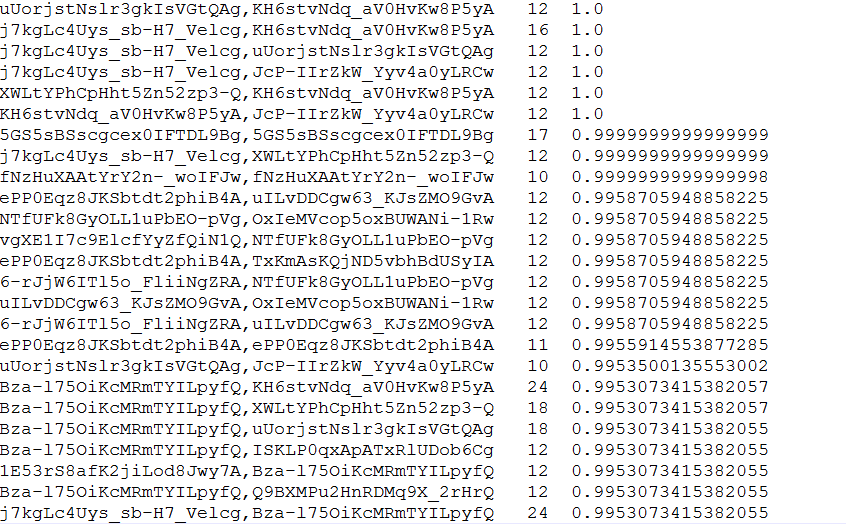
Input – > (business ID, business ID), ([(rating,rating)(rating,rating)…]

Output –> (business ID, business ID), (Similarity Score, Number of users who rated both)



**Final Output of Step 2:**

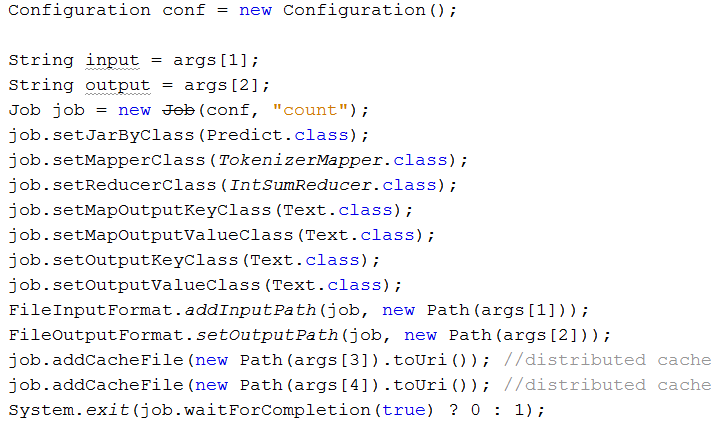
**BuisnessID,BusinessID Number Similarity Score**



**Graphical Analysis Report for Recommendation Analysis**

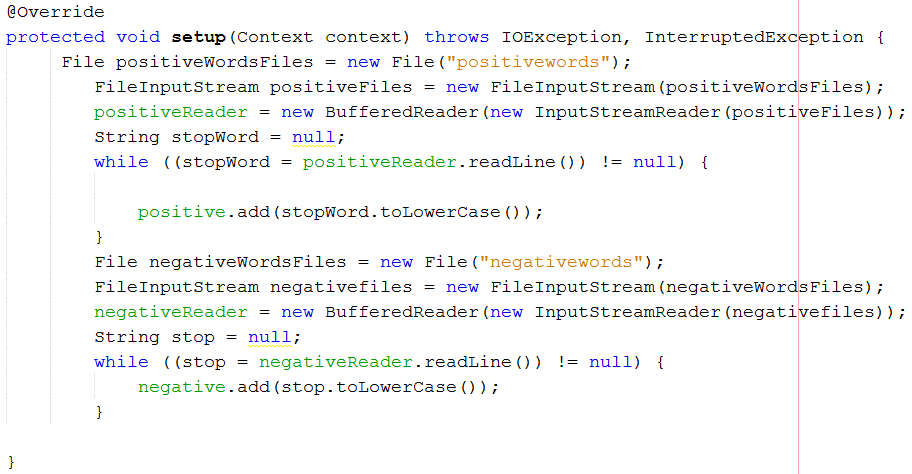
1. **Sentiment Analysis Algorithm:** Implemented Sentiments analysis and calculated average rating received by business ID’s in particular month. From the analysis we can predict that in which month the sentiment reviews and ratings received by businesses were good or bad which can help to understand behavior of user visiting restaurants in particular period. Used **Distributed cache** to store the files containing positive and negative words need for doing sentiment analysis of review given by user.

**Driver function**

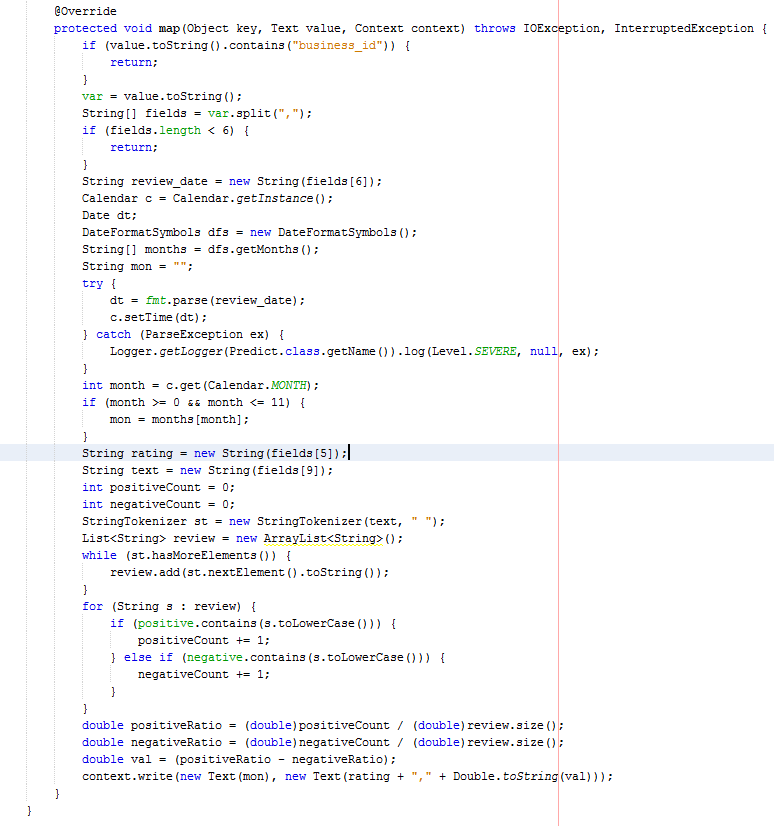


**Mapper function:**

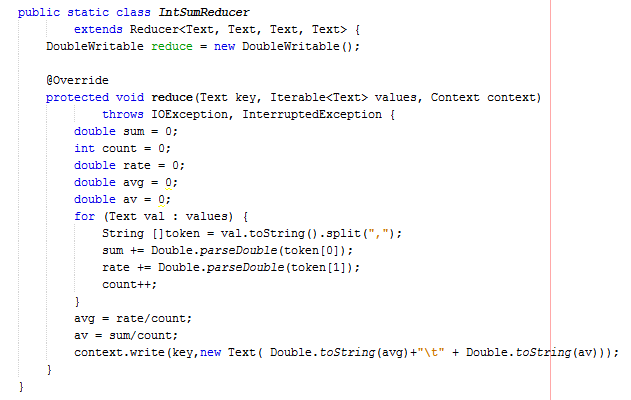
**Setup method to read the cache files**



**Map function:**

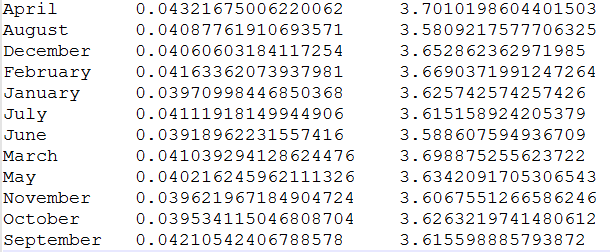


**Reduce Function:**



**Final Output of analysis**

**Month Sentiment Score Average Rating**



**Graphical Analysis Report:**

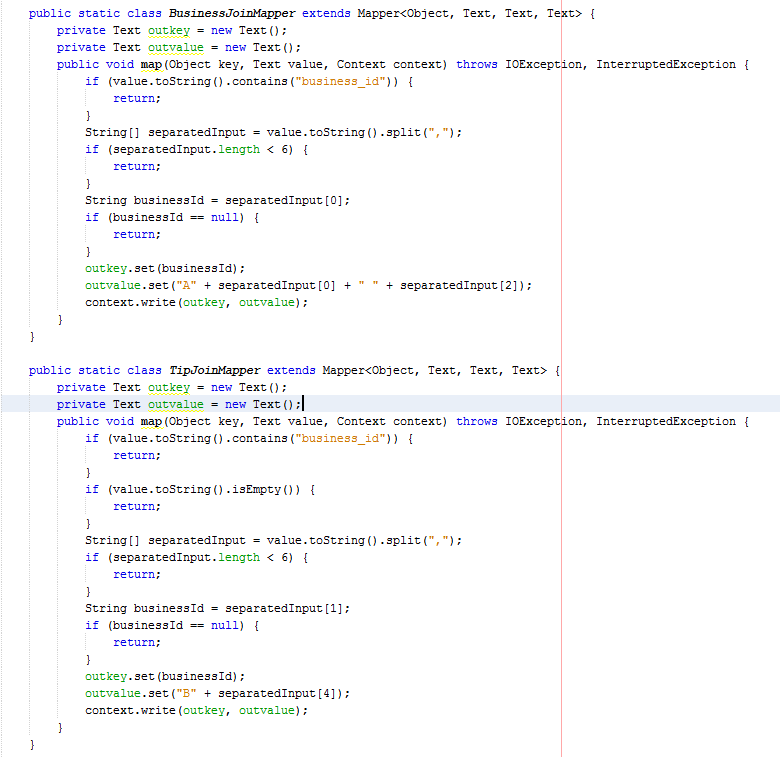
1. **Reduce Side Inner Join:** Joined two datasets on businessID’s to find the city having businesses with most user reviews. In the first job two datasets were taken as input and joined the two datasets whose output was used as input by second job to find the aggregation count of user reviews per place. In the third job the key and value were interchanged and SortComparator was used to do descending order comparison and generate top 25 cities as output. This analysis can tell us that which particular city has good restaurants, active users and this data can be used by both users to visit a city and business person to setup a business in particular city.

**Driver function:**

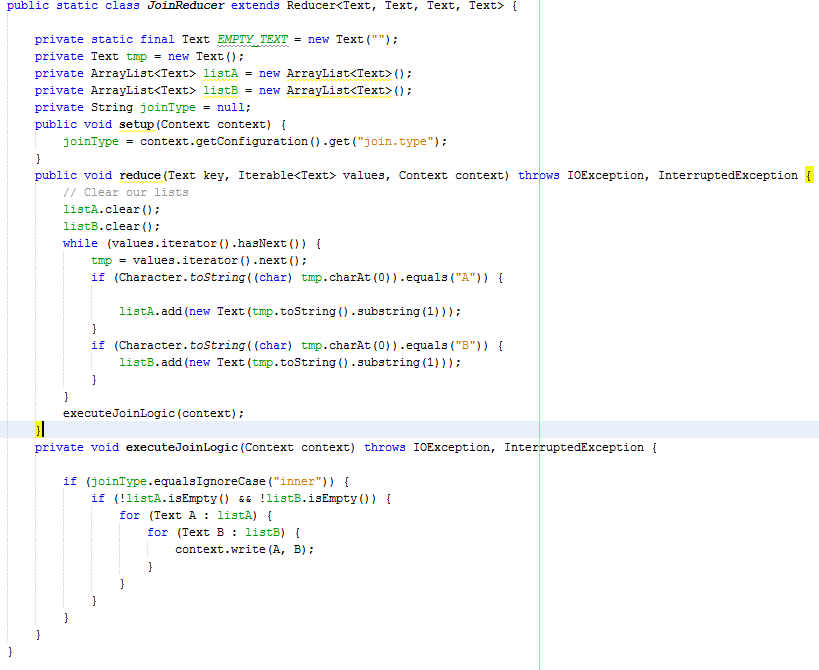


**Job 1: -**

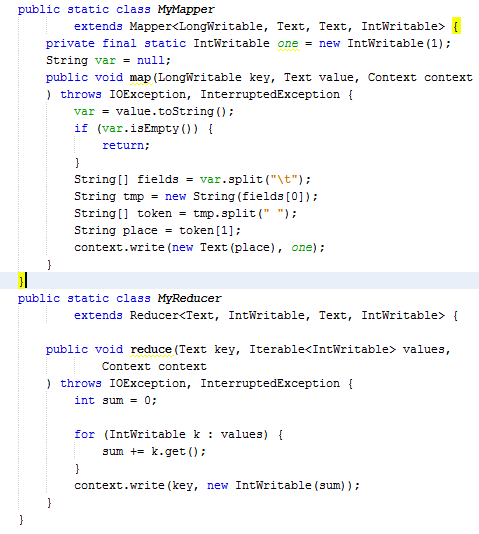
**Mapper function**



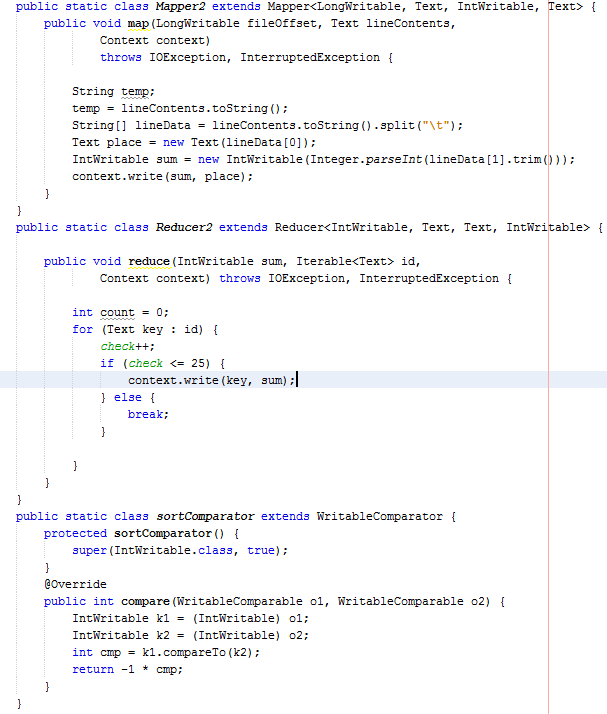
**Reducer Function and Join logic**



**Job 2: - Mapper and Reducer function**



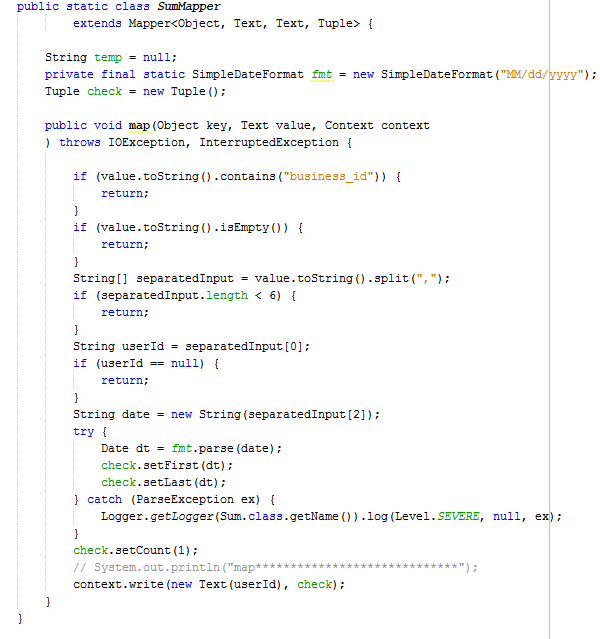
**Job 3: - Mapper, Reducer and SortComparator Fucntion**



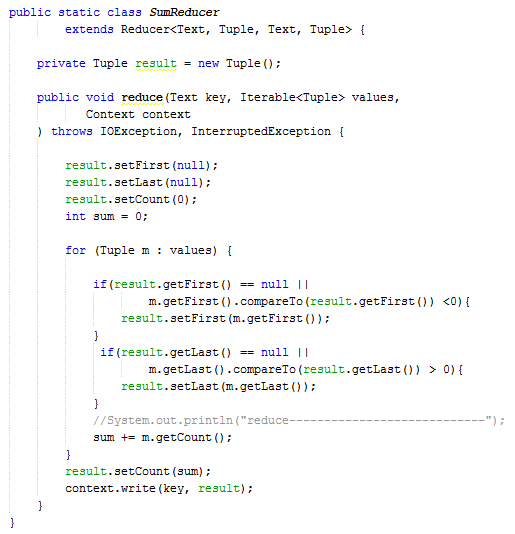
**Graphical Analysis Report**

1. **Summarization pattern for finding the number of reviews by each user and his first and last review date.**

**Mapper Function:**

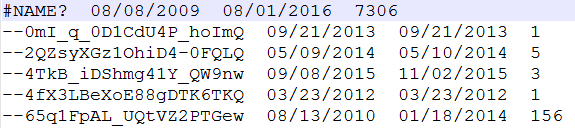


**Reducer Function:**



**Final sample output:**

**Userid First date of review Last date of review Number of reviews**



**Appendix**

1. **Simple Random Sampling:**

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package filter;

import java.io.IOException;

import java.util.Random;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.NullWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

/\*\*

\*

\* @author anike

\*/

public class Filter {

public static class TokenizerMapper

extends Mapper<Object, Text, NullWritable, Text> {

private Random randomPercent = new Random();

private float percentage;

@Override

protected void setup(Context context) throws IOException, InterruptedException {

percentage = .80f;

}

public void map(Object key, Text value, Mapper.Context context

) throws IOException, InterruptedException {

if(randomPercent.nextDouble() < percentage){

context.write(NullWritable.get(),value);

}

}

}

public static void main(String[] args) throws IOException, InterruptedException, ClassNotFoundException {

// TODO code application logic here

Configuration conf = new Configuration();

Job job = new Job(conf, "filter");

job.setJarByClass(Filter.class);

job.setMapperClass(TokenizerMapper.class);

job.setOutputKeyClass(NullWritable.class);

job.setOutputValueClass(Text.class);

job.setNumReduceTasks(1);

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

FileOutputFormat.setOutputPath(job, new Path(args[2]));

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

}

}

1. **Partitioning (Data Organization Pattern)**

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package partition;

import java.io.IOException;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Calendar;

import java.util.Date;

import java.util.logging.Level;

import java.util.logging.Logger;

import org.apache.hadoop.conf.Configurable;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.NullWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Partitioner;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

/\*\*

\*

\* @author anike

\*/

public class Partition {

public static class PartitionMapper

extends Mapper<Object, Text, IntWritable, Text> {

String temp = null;

private final static SimpleDateFormat fmt = new SimpleDateFormat("MM/dd/yyyy");

private int year = 0;

public void map(Object key, Text value, Mapper.Context context

) throws IOException, InterruptedException {

if (value.toString().contains("business\_id")) {

return;

}

if (value.toString().isEmpty()) {

return;

}

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

if (separatedInput.length < 6) {

return;

}

String date = new String(separatedInput[2]);

try {

Calendar c = Calendar.getInstance();

Date dt = fmt.parse(date);

c.setTime(dt);

year = c.get(Calendar.YEAR);

} catch (ParseException ex) {

Logger.getLogger(Partition.class.getName()).log(Level.SEVERE, null, ex);

}

// System.out.println("map\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

context.write(new IntWritable(year), value);

}

}

public static class YearPartitoner extends Partitioner<IntWritable, Text>

implements Configurable {

private Configuration conf = null;

private int year = 0;

private static final String DATE\_YEAR = "date.year";

@Override

public int getPartition(IntWritable key, Text value, int i) {

return key.get() - year;

}

@Override

public void setConf(Configuration conf) {

this.conf = conf;

year = conf.getInt(DATE\_YEAR, 0);

}

@Override

public Configuration getConf() {

return conf;

}

public static void setAccessYear(Job job, int accessYear) {

job.getConfiguration().setInt(DATE\_YEAR, accessYear);

}

}

public static class SumReducer

extends Reducer<IntWritable, Text, Text, NullWritable> {

public void reduce(IntWritable key, Iterable<Text> values,

Reducer.Context context

) throws IOException, InterruptedException {

for(Text t :values){

context.write(t, NullWritable.get());

}

}

}

public static void main(String[] args) throws IOException, InterruptedException, ClassNotFoundException {

Configuration conf = new Configuration();

Job job = new Job(conf, "Partition");

job.setJarByClass(Partition.class);

job.setMapperClass(PartitionMapper.class);

job.setInputFormatClass(TextInputFormat.class);

job.setPartitionerClass(YearPartitoner.class);

YearPartitoner.setAccessYear(job, 2009);

job.setNumReduceTasks(8);

job.setMapOutputKeyClass(IntWritable.class);

job.setMapOutputValueClass(Text.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(NullWritable.class);

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

FileOutputFormat.setOutputPath(job, new Path(args[2]));

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

}

}

1. **Recommendation Algorithm**

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package reco;

import java.io.IOException;

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.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

/\*\*

\*

\* @author anike

\*/

public class Reco {

public static class MedianMapper

extends Mapper<Object, Text, Text, Tuple> {

String temp = null;

Tuple check = new Tuple();

public void map(Object key, Text value, Mapper.Context context

) throws IOException, InterruptedException {

if (value.toString().contains("business\_id")) {

return;

}

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

if (separatedInput.length < 9) {

return;

}

String businessId = separatedInput[3];

if (businessId == null) {

return;

}

String user\_id = new String(separatedInput[0]);

String business\_rating = new String(separatedInput[5]);

double rating = Double.parseDouble(business\_rating.trim());

check.setRating(rating);

check.setBusinessid(businessId);

context.write(new Text(user\_id), check);

}

}

public static class MedianReducer

extends Reducer<Text, Tuple, Text, Text> {

public void reduce(Text key, Iterable<Tuple> values,

Context context

) throws IOException, InterruptedException {

long count = 0;

String t = "";

for (Tuple val : values) {

t = t.concat(val.getBusinessid()+ "," + val.getRating() + " ");

}

context.write(key, new Text(t.trim()));

}

}

public static void main(String[] args) throws IOException, InterruptedException, ClassNotFoundException {

Configuration conf = new Configuration();

String input = args[1];

String output = args[2];

Job job = new Job(conf, "Reco");

job.setJarByClass(Reco.class);

job.setMapperClass(MedianMapper.class);

job.setReducerClass(MedianReducer.class);

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(Tuple.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(Text.class);

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

FileOutputFormat.setOutputPath(job, new Path(args[2]));

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

}

}

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package reco;

import java.io.DataInput;

import java.io.DataOutput;

import java.io.IOException;

import org.apache.hadoop.io.Writable;

import org.apache.hadoop.io.WritableUtils;

/\*\*

\*

\* @author anike

\*/

public class Tuple implements Writable {

private String businessid;

private double rating;

public String getBusinessid() {

return businessid;

}

public void setBusinessid(String businessid) {

this.businessid = businessid;

}

public double getRating() {

return rating;

}

public void setRating(double rating) {

this.rating = rating;

}

@Override

public void readFields(DataInput in) throws IOException {

businessid = WritableUtils.readString(in);

rating = in.readDouble();

}

@Override

public void write(DataOutput out) throws IOException {

WritableUtils.writeString(out, businessid);

out.writeDouble(rating);

}

@Override

public String toString() {

return businessid + "\t" + rating;

}

}

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package userratedcount;

import java.io.IOException;

import java.util.Collections;

import java.util.Comparator;

import java.util.HashMap;

import java.util.LinkedHashMap;

import java.util.LinkedList;

import java.util.List;

import java.util.Map;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.DoubleWritable;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

/\*\*

\*

\* @author anike

\*/

public class UserRatedCount {

public static class TokenizerMapper

extends Mapper<Object, Text, Text, Text> {

String temp = null;

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

private final Text mapKey = new Text();

public void map(Object key, Text value, Context context

) throws IOException, InterruptedException {

temp = value.toString();

if (temp.isEmpty()) {

return;

}

String[] fields = temp.split("\t");

String movie\_id = new String(fields[1]);

String[] tokens = movie\_id.toString().split(" ");

if (tokens.length >= 2) {

for (int i = 0; i < tokens.length; i++) {

for (int j = i + 1; j < tokens.length; j++) {

String groupings = tokens[i] + "," + tokens[j]; // Add 2 movies with ",". For e.g. "19,2,21,1"

String[] moviesAndRatings = groupings.split(",");

if (moviesAndRatings.length == 4) {

context.write(new Text(moviesAndRatings[0] + "," + moviesAndRatings[2]),

new Text(moviesAndRatings[1] + "," + moviesAndRatings[3]));

}

}

}

}

}

}

public static class IntSumReducer

extends Reducer<Text, Text, Text, Text> {

private Map<Text, DoubleWritable> topCountMap = new HashMap<Text, DoubleWritable>();

public void reduce(Text key, Iterable<Text> values,

Context context

) throws IOException, InterruptedException {

double sum\_aa = 0;

double sum\_bb =0;

double sum\_ab = 0;

int count = 0;

double score = 0;

double num =0;

double den = 0;

for (Text val : values) {

String [] token = val.toString().split(",");

double aa = Double.parseDouble(token[0]);

double bb = Double.parseDouble(token[1]);

sum\_ab += aa \* bb;

sum\_aa += aa \* aa;

sum\_bb += bb \* bb;

count ++;

}

num = sum\_ab;

den = Math.sqrt(sum\_aa) \*Math.sqrt(sum\_bb);

if(den != 0){

score = num/den;

}

//String p = "The number of movie rated by"+ key.toString();

if(count >9){

topCountMap.put(new Text(key +"\t"+ Integer.toString(count)), new DoubleWritable(score));

//context.write(key, new Text(Double.toString(score) + "\t" + Integer.toString(count)));

}

// context.write(key, new Text(Integer.toString(count)));

}

@Override

protected void cleanup(Context context) throws IOException, InterruptedException {

Map<Text, DoubleWritable> sortedValues = sortByValues(topCountMap);

int count = 0;

for(Text key: sortedValues.keySet()){

if(count++ == 25){

break;

}

context.write(key, new Text(sortedValues.get(key).toString()));

}

}

private static Map<Text, DoubleWritable> sortByValues(Map<Text, DoubleWritable> map) {

List<Map.Entry<Text, DoubleWritable>> entries = new LinkedList<Map.Entry<Text, DoubleWritable>>(map.entrySet());

Collections.sort(entries, new Comparator<Map.Entry<Text, DoubleWritable>>() {

public int compare(Map.Entry<Text, DoubleWritable> o1, Map.Entry<Text, DoubleWritable> o2) {

return o2.getValue().compareTo(o1.getValue());

}

});

Map<Text, DoubleWritable> sortedMap = new LinkedHashMap<Text, DoubleWritable>();

for (Map.Entry<Text, DoubleWritable> entry : entries) {

sortedMap.put(entry.getKey(), entry.getValue());

}

return sortedMap;

}

}

/\*\*

\* @param args the command line arguments

\*/

public static void main(String[] args) throws IOException, InterruptedException, ClassNotFoundException {

Configuration conf = new Configuration();

String input = args[1];

String output = args[2];

Job job = new Job(conf, "User Count");

job.setJarByClass(UserRatedCount.class);

job.setMapperClass(TokenizerMapper.class);

job.setReducerClass(IntSumReducer.class);

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(Text.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(Text.class);

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

FileOutputFormat.setOutputPath(job, new Path(args[2]));

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

}

}

1. **Sentiment Analysis Algorithm:**

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package predict;

import java.io.BufferedReader;

import java.io.File;

import java.io.FileInputStream;

import java.io.IOException;

import java.io.InputStreamReader;

import java.text.DateFormatSymbols;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.ArrayList;

import java.util.Calendar;

import java.util.Date;

import java.util.HashSet;

import java.util.List;

import java.util.Set;

import java.util.StringTokenizer;

import java.util.logging.Level;

import java.util.logging.Logger;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.DoubleWritable;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

/\*\*

\*

\* @author anike

\*/

public class Predict {

public static class TokenizerMapper

extends Mapper<Object, Text, Text, Text> {

Set<String> positive = new HashSet<String>();

Set<String> negative = new HashSet<String>();

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

String var = null;

private final static SimpleDateFormat fmt = new SimpleDateFormat("MM/dd/yyyy");

private BufferedReader positiveReader;

private BufferedReader negativeReader;

@Override

protected void setup(Context context) throws IOException, InterruptedException {

File positiveWordsFiles = new File("positivewords");

FileInputStream positiveFiles = new FileInputStream(positiveWordsFiles);

positiveReader = new BufferedReader(new InputStreamReader(positiveFiles));

String stopWord = null;

while ((stopWord = positiveReader.readLine()) != null) {

positive.add(stopWord.toLowerCase());

}

File negativeWordsFiles = new File("negativewords");

FileInputStream negativefiles = new FileInputStream(negativeWordsFiles);

negativeReader = new BufferedReader(new InputStreamReader(negativefiles));

String stop = null;

while ((stop = negativeReader.readLine()) != null) {

negative.add(stop.toLowerCase());

}

}

@Override

protected void map(Object key, Text value, Context context) throws IOException, InterruptedException {

if (value.toString().contains("business\_id")) {

return;

}

var = value.toString();

String[] fields = var.split(",");

if (fields.length < 6) {

return;

}

String review\_date = new String(fields[6]);

Calendar c = Calendar.getInstance();

Date dt;

DateFormatSymbols dfs = new DateFormatSymbols();

String[] months = dfs.getMonths();

String mon = "";

try {

dt = fmt.parse(review\_date);

c.setTime(dt);

} catch (ParseException ex) {

Logger.getLogger(Predict.class.getName()).log(Level.SEVERE, null, ex);

}

int month = c.get(Calendar.MONTH);

if (month >= 0 && month <= 11) {

mon = months[month];

}

String rating = new String(fields[5]);

String text = new String(fields[9]);

int positiveCount = 0;

int negativeCount = 0;

StringTokenizer st = new StringTokenizer(text, " ");

List<String> review = new ArrayList<String>();

while (st.hasMoreElements()) {

review.add(st.nextElement().toString());

}

for (String s : review) {

if (positive.contains(s.toLowerCase())) {

positiveCount += 1;

} else if (negative.contains(s.toLowerCase())) {

negativeCount += 1;

}

}

double positiveRatio = (double)positiveCount / (double)review.size();

double negativeRatio = (double)negativeCount / (double)review.size();

double val = (positiveRatio - negativeRatio);

context.write(new Text(mon), new Text(rating + "," + Double.toString(val)));

}

}

public static class IntSumReducer

extends Reducer<Text, Text, Text, Text> {

DoubleWritable reduce = new DoubleWritable();

@Override

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

throws IOException, InterruptedException {

double sum = 0;

int count = 0;

double rate = 0;

double avg = 0;

double av = 0;

for (Text val : values) {

String []token = val.toString().split(",");

sum += Double.parseDouble(token[0]);

rate += Double.parseDouble(token[1]);

count++;

}

avg = rate/count;

av = sum/count;

context.write(key,new Text( Double.toString(avg)+"\t" + Double.toString(av)));

}

}

public static void main(String[] args) throws IOException, InterruptedException, ClassNotFoundException {

Configuration conf = new Configuration();

String input = args[1];

String output = args[2];

Job job = new Job(conf, "count");

job.setJarByClass(Predict.class);

job.setMapperClass(TokenizerMapper.class);

job.setReducerClass(IntSumReducer.class);

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(Text.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(Text.class);

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

FileOutputFormat.setOutputPath(job, new Path(args[2]));

job.addCacheFile(new Path(args[3]).toUri()); //distributed cache

job.addCacheFile(new Path(args[4]).toUri()); //distributed cache

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

}

}

1. **Reduce side Inner Join:**

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package join;

import java.io.IOException;

import java.util.ArrayList;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.io.WritableComparable;

import org.apache.hadoop.io.WritableComparator;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.input.MultipleInputs;

import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

/\*\*

\*

\* @author anike

\*/

public class Join {

static int check = 0;

public static class BusinessJoinMapper extends Mapper<Object, Text, Text, Text> {

private Text outkey = new Text();

private Text outvalue = new Text();

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

if (value.toString().contains("business\_id")) {

return;

}

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

if (separatedInput.length < 6) {

return;

}

String businessId = separatedInput[0];

if (businessId == null) {

return;

}

outkey.set(businessId);

outvalue.set("A" + separatedInput[0] + " " + separatedInput[2]);

context.write(outkey, outvalue);

}

}

public static class TipJoinMapper extends Mapper<Object, Text, Text, Text> {

private Text outkey = new Text();

private Text outvalue = new Text();

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

if (value.toString().contains("business\_id")) {

return;

}

if (value.toString().isEmpty()) {

return;

}

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

if (separatedInput.length < 6) {

return;

}

String businessId = separatedInput[1];

if (businessId == null) {

return;

}

outkey.set(businessId);

outvalue.set("B" + separatedInput[4]);

context.write(outkey, outvalue);

}

}

public static class JoinReducer extends Reducer<Text, Text, Text, Text> {

private static final Text EMPTY\_TEXT = new Text("");

private Text tmp = new Text();

private ArrayList<Text> listA = new ArrayList<Text>();

private ArrayList<Text> listB = new ArrayList<Text>();

private String joinType = null;

public void setup(Context context) {

joinType = context.getConfiguration().get("join.type");

}

public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {

// Clear our lists

listA.clear();

listB.clear();

while (values.iterator().hasNext()) {

tmp = values.iterator().next();

if (Character.toString((char) tmp.charAt(0)).equals("A")) {

listA.add(new Text(tmp.toString().substring(1)));

}

if (Character.toString((char) tmp.charAt(0)).equals("B")) {

listB.add(new Text(tmp.toString().substring(1)));

}

}

executeJoinLogic(context);

}

private void executeJoinLogic(Context context) throws IOException, InterruptedException {

if (joinType.equalsIgnoreCase("inner")) {

if (!listA.isEmpty() && !listB.isEmpty()) {

for (Text A : listA) {

for (Text B : listB) {

context.write(A, B);

}

}

}

}

}

}

public static class MyMapper

extends Mapper<LongWritable, Text, Text, IntWritable> {

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

String var = null;

public void map(LongWritable key, Text value, Context context

) throws IOException, InterruptedException {

var = value.toString();

if (var.isEmpty()) {

return;

}

String[] fields = var.split("\t");

String tmp = new String(fields[0]);

String[] token = tmp.split(" ");

String place = token[1];

context.write(new Text(place), one);

}

}

public static class MyReducer

extends Reducer<Text, IntWritable, Text, IntWritable> {

public void reduce(Text key, Iterable<IntWritable> values,

Context context

) throws IOException, InterruptedException {

int sum = 0;

for (IntWritable k : values) {

sum += k.get();

}

context.write(key, new IntWritable(sum));

}

}

public static class Mapper2 extends Mapper<LongWritable, Text, IntWritable, Text> {

public void map(LongWritable fileOffset, Text lineContents,

Context context)

throws IOException, InterruptedException {

String temp;

temp = lineContents.toString();

String[] lineData = lineContents.toString().split("\t");

Text place = new Text(lineData[0]);

IntWritable sum = new IntWritable(Integer.parseInt(lineData[1].trim()));

context.write(sum, place);

}

}

public static class Reducer2 extends Reducer<IntWritable, Text, Text, IntWritable> {

public void reduce(IntWritable sum, Iterable<Text> id,

Context context) throws IOException, InterruptedException {

int count = 0;

for (Text key : id) {

check++;

if (check <= 25) {

context.write(key, sum);

} else {

//System.out.println("break");

break;

}

}

}

}

public static class sortComparator extends WritableComparator {

protected sortComparator() {

super(IntWritable.class, true);

}

@Override

public int compare(WritableComparable o1, WritableComparable o2) {

IntWritable k1 = (IntWritable) o1;

IntWritable k2 = (IntWritable) o2;

int cmp = k1.compareTo(k2);

return -1 \* cmp;

}

}

public static int main(String[] args) throws IOException, InterruptedException, ClassNotFoundException {

Configuration conf = new Configuration();

Job job = new Job(conf, "ReduceSideJoin");

job.setJarByClass(Join.class);

MultipleInputs.addInputPath(job, new Path(args[0]), TextInputFormat.class, BusinessJoinMapper.class);

MultipleInputs.addInputPath(job, new Path(args[1]), TextInputFormat.class, TipJoinMapper.class);

job.getConfiguration().set("join.type", "inner");

job.setReducerClass(JoinReducer.class);

job.setOutputFormatClass(TextOutputFormat.class);

TextOutputFormat.setOutputPath(job, new Path(args[2]));

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(Text.class);

boolean success = job.waitForCompletion(true);

if (success) {

Job job2 = new Job(conf, "Job 2");

job2.setJarByClass(Join.class);

job2.setMapperClass(MyMapper.class);

job2.setReducerClass(MyReducer.class);

job2.setMapOutputKeyClass(Text.class);

job2.setMapOutputValueClass(IntWritable.class);

job2.setOutputKeyClass(Text.class);

job2.setOutputValueClass(IntWritable.class);

FileInputFormat.addInputPath(job2, new Path(args[2]));

FileOutputFormat.setOutputPath(job2, new Path(args[3]));

boolean flag = job2.waitForCompletion(true);

if (flag) {

Job job3 = Job.getInstance(conf, "Sorting");

job3.setJarByClass(Join.class);

job3.setMapperClass(Mapper2.class);

job3.setReducerClass(Reducer2.class);

job3.setSortComparatorClass(sortComparator.class);

job3.setMapOutputKeyClass(IntWritable.class);

job3.setMapOutputValueClass(Text.class);

job3.setOutputKeyClass(Text.class);

job3.setOutputValueClass(IntWritable.class);

FileInputFormat.addInputPath(job3, new Path(args[3]));

FileOutputFormat.setOutputPath(job3, new Path(args[4]));

System.exit(job3.waitForCompletion(true) ? 0 : 1);

}

}

return 0;

}

}

1. **Summarization pattern to find user review count and his first and last date of review**

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package sum;

import java.io.IOException;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Date;

import java.util.logging.Level;

import java.util.logging.Logger;

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.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

/\*\*

\*

\* @author anike

\*/

public class Sum {

public static class SumMapper

extends Mapper<Object, Text, Text, Tuple> {

String temp = null;

private final static SimpleDateFormat fmt = new SimpleDateFormat("MM/dd/yyyy");

Tuple check = new Tuple();

public void map(Object key, Text value, Context context

) throws IOException, InterruptedException {

if (value.toString().contains("business\_id")) {

return;

}

if (value.toString().isEmpty()) {

return;

}

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

if (separatedInput.length < 6) {

return;

}

String userId = separatedInput[0];

if (userId == null) {

return;

}

String date = new String(separatedInput[2]);

try {

Date dt = fmt.parse(date);

check.setFirst(dt);

check.setLast(dt);

} catch (ParseException ex) {

Logger.getLogger(Sum.class.getName()).log(Level.SEVERE, null, ex);

}

check.setCount(1);

// System.out.println("map\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

context.write(new Text(userId), check);

}

}

public static class SumReducer

extends Reducer<Text, Tuple, Text, Tuple> {

private Tuple result = new Tuple();

public void reduce(Text key, Iterable<Tuple> values,

Context context

) throws IOException, InterruptedException {

result.setFirst(null);

result.setLast(null);

result.setCount(0);

int sum = 0;

for (Tuple m : values) {

if(result.getFirst() == null ||

m.getFirst().compareTo(result.getFirst()) <0){

result.setFirst(m.getFirst());

}

if(result.getLast() == null ||

m.getLast().compareTo(result.getLast()) > 0){

result.setLast(m.getLast());

}

//System.out.println("reduce----------------------------");

sum += m.getCount();

}

result.setCount(sum);

context.write(key, result);

}

}

public static void main(String[] args) throws IOException, InterruptedException, ClassNotFoundException {

Configuration conf = new Configuration();

Job job = new Job(conf, "Sum");

job.setJarByClass(Sum.class);

job.setMapperClass(SumMapper.class);

job.setInputFormatClass(TextInputFormat.class);

job.setReducerClass(SumReducer.class);

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(Tuple.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(Tuple.class);

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

FileOutputFormat.setOutputPath(job, new Path(args[2]));

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

}

}

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package sum;

import java.io.DataInput;

import java.io.DataOutput;

import java.io.IOException;

import java.text.SimpleDateFormat;

import java.util.Date;

import org.apache.hadoop.io.Writable;

/\*\*

\*

\* @author anike

\*/

public class Tuple implements Writable {

private Date first = new Date();

private Date last = new Date();

private long count = 0;

private SimpleDateFormat fmt = new SimpleDateFormat("MM/dd/yyyy");

public Date getFirst() {

return first;

}

public void setFirst(Date first) {

this.first = first;

}

public Date getLast() {

return last;

}

public void setLast(Date last) {

this.last = last;

}

public long getCount() {

return count;

}

public void setCount(long count) {

this.count = count;

}

@Override

public String toString() {

if((first != null) && (last !=null)){

return fmt.format(first) + "\t" + fmt.format(last) + "\t" + count;

}

return "";

}

@Override

public void readFields(DataInput in) throws IOException {

first = new Date(in.readLong());

last = new Date(in.readLong());

count = in.readLong();

}

@Override

public void write(DataOutput out) throws IOException {

out.writeLong(first.getTime());

out.writeLong(last.getTime());

out.writeLong(count);

}

}