

Social Network of YouTube Videos

Advanced Database Management Systems

By : Karjout abdeslam & fayz sabir

Encadre par Mr tkatek

# **Énoncé du problème**

1. Analyse de l’ensemble de données de réseaux sociaux « YouTube » basé sur différents champs de colonnes.
2. Recommandation de vidéos YouTube à un utilisateur en fonction de son évaluation
3. Recommandation des abonnés

# **Résumé**

L’ensemble de données utilisé dans ce projet est fourni par l’Université SimonFraser. L’ensemble de données comporte cinq fichiers différents basés sur les données collectées par le robot.

Le jeu de données était disponible sur l’URL suivante :

<http://netsg.cs.sfu.ca/youtubedata/>

Pour la recommandation d’amis et la recommandation vidéo, nous avons pris l’ensemble de données suivant pour notre analyse:

<http://datasets.syr.edu/datasets/YouTube2.html>

Nous enregistrons les informations suivantes d’une vidéo YouTube dans l’ordre; ils sont divisés par '\t’dans le fichier de données.

|  |  |
| --- | --- |
| video ID | an 11-digit string, which is unique |
| uploader | a string of the video uploader's username |
| age | an integer number of days between the date when the video was uploaded and Feb.15, 2007 (YouTube's establishment) |
| category | a string of the video category chosen by the uploader |
| length | an integer number of the video length |
| views | an integer number of the views |
| rate | a float number of the video rate |
| ratings | an integer number of the ratings |
| comments | an integer number of the comments |
| related IDs | up to 20 strings of the related video IDs |

**Ensembles de données d’informations utilisateur**

Nous avons collecté les informations sur les utilisateurs de YouTube. Le robot récupère des informations sur le nombre de vidéos téléchargées et d’amis de chaque utilisateur à partir de l’API YouTube, pour un total de plus de 1 million d’utilisateurs. Il y a « user.txt », contenant les informations du nombre de téléchargements, de montres et d’amis dans l’ordre.

2 fichiers sont inclus :

1. nodes.csv

-- c’est le fichier de tous les utilisateurs. Ce fichier fonctionne comme un dictionnaire de tous les utilisateurs de cet ensemble de données. C’est utile pour une référence rapide. Il contient tous les ID de nœud utilisés dans le jeu de données

2. edges.csv

- c’est le réseau d’amitié entre les utilisateurs. Les amis de l’utilisateur sont représentés à l’aide d’arêtes.

Comme le réseau est symétrique, chaque arête n’est représentée qu’une seule fois. Voici un exemple.

1,2

Cela signifie que l’utilisateur avec l’ID « 1 » est ami avec l’ID utilisateur « 2 ».

Tous les fichiers sources nécessitaient beaucoup de nettoyage de données un pré-traitement d qui a été fait en Python. Après ces étapes, les fichiers csv ont été utilisés en effectuant différentes analyses à l’aide du paradigme informatique MapReduce qui est utilisé populairement pour travailler avec un volume élevé de données. JAVA était le langage principal utilisé pour écrire tous les programmes MapReduce et implémenter ses différents modèles de conception tels que :

1. Filtrage
2. Modèles de jointure
3. Organisation des données
4. Résumé

La recommandation de vidéos aux utilisateurs a également été faite en utilisant la technique de recommandation basée sur le contenu en utilisant Mahout.

Certaines des analyses effectuées sur l’ensemble de données constitué de données de restaurants aux États-Unis sont:

1. Note maximale Note totale et nombre total de commentaires par ID vidéo
2. Moyenne des cotes mobiles par Video\_ID
3. Meilleur Youtuber basé sur les vidéos téléchargées
4. Top 50 des vidéos YouTube préférées
5. Nombre total de vidéos YouTube par catégorie
6. Triage par catégories
7. Résultat de chaînage sur triage pour obtenir les 25 meilleures vidéos par catégorie
8. Recommandation d’abonnés basée sur des abonnés connectés
9. Recommandation basée sur la vidéo en utilisant Mahout.
10. Nombre total de vues en fonction de l’ID vidéo

# **Description de l’analyse**

**Moyenne de la note mobile par Video\_ID.**

Cette analyse a consisté à calculer la note moyenne des vidéos téléchargées par les utilisateurs et la note fournie sur les vidéos par différents utilisateurs.

**Note maximale Note totale et nombre total de commentaires par ID vidéo**

Cette analyse visait à analyser le nombre total de commentaires, la note totale et la note maximale fournis à une vidéo YouTube sur la base desquels nous pouvons déterminer quels films doivent être supprimés à l’avenir.

**Meilleur Youtuber basé sur les vidéos téléchargées**

Cette analyse représente le meilleur YouTuber qui a contribué à la plupart des vidéos sur YouTube.

**Top 50 des vidéos YouTube préférées**

**Cette analyse aide à trouver les 50 meilleures vidéos YouTube.**

**Nombre total de vidéos YouTube par catégorie**

Cette analyse permet de trouver le total des vidéos en fonction des catégories.

**Binning par catégories**

**Séparez les vidéos en fonction des catégories.**

**Résultat de chaînage sur Binning pour obtenir les 25 meilleures vidéos par catégorie**

**Nous obtenons toutes les 25 meilleures vidéos en fonction de la catégorie.**

**Recommandation d’abonnés basée sur des abonnés connectés**

**Pour augmenter le nombre d’abonnés, nous avons créé une recommandation où nous pouvons suggérer à un utilisateur de suivre l’ensemble des utilisateurs.**

**Recommandation basée sur la vidéo en utilisant Mahout.**

**Utilisé Mahout Library pour trouver les vidéos aimées par l’utilisateur et les recommander en fonction de la similitude de la note.**

**Nombre total de vues en fonction de l’ID vidéo**

**Nombre total de vues qu’une vidéo a obtenues des utilisateurs.**

**Code:**

**Note maximale et nombre total de commentaires par ID vidéo**

**/\***

**\* 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 rating\_summerization;**

**import java.io.DataInput;**

**import java.io.DataOutput;**

**import java.io.IOException;**

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

**/\*\***

**\***

**\* @author Karjout**

**\*/**

**public class MinMaxCountTuple implements Writable {**

**private float averageRating;**

**private float totalRating;**

**private float totalComment;**

**public float getAverageRating() {**

**return averageRating;**

**}**

**public void setAverageRating(float averageRating) {**

**this.averageRating = averageRating;**

**}**

**public float getTotalRating() {**

**return totalRating;**

**}**

**public void setTotalRating(float totalRating) {**

**this.totalRating = totalRating;**

**}**

**public float getTotalComment() {**

**return totalComment;**

**}**

**public void setTotalComment(float totalComment) {**

**this.totalComment = totalComment;**

**}**

**@Override**

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

**d.writeFloat(averageRating);**

**d.writeFloat(totalRating);**

**d.writeFloat(totalComment);**

**}**

**@Override**

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

**averageRating = di.readFloat();**

**totalRating = di.readFloat();**

**totalComment = di.readFloat();**

**}**

**@Override**

**public String toString() {**

**return (averageRating + "\t" + totalRating + "\t" + totalComment);**

**}**

**}**

**package rating\_summerization;**

**import java.io.IOException;**

**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.lib.input.FileInputFormat;**

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

**/\*\***

**\***

**\* @author Karjout**

**\*/**

**public class Rating\_Summerization {**

**/\*\***

**\* @param args the command line arguments**

**\*/**

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

**try {**

**Configuration conf = new Configuration();**

**Job job = Job.getInstance(conf, "VideoMinMaxRating");**

**job.setJarByClass(Rating\_Summerization.class);**

**job.setMapperClass(Summ\_Mapper.class);**

**job.setCombinerClass(Summ\_Reducer.class);**

**job.setReducerClass(Summ\_Reducer.class);**

**job.setOutputKeyClass(Text.class);**

**job.setOutputValueClass(MinMaxCountTuple.class);**

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

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

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

**} catch (InterruptedException | ClassNotFoundException ex) {**

**Logger.getLogger(Rating\_Summerization.class.getName()).log(Level.SEVERE, null, ex);**

**}**

**}**

**}**

**/\***

**\* 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 rating\_summerization;**

**import java.io.IOException;**

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

**import org.apache.hadoop.mapreduce.Mapper;**

**/\*\***

**\***

**\* @author Karjout**

**\*/**

**class Summ\_Mapper extends Mapper<Object, Text, Text, MinMaxCountTuple> {**

**private Text video\_ID = new Text();**

**private MinMaxCountTuple outTuple = new MinMaxCountTuple();**

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

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

**video\_ID.set(input[0]);**

**outTuple.setTotalRating(Float.valueOf(input[7]));**

**outTuple.setAverageRating(Float.valueOf(input[6]));**

**outTuple.setTotalComment(Float.valueOf(input[8]));**

**context.write(video\_ID, outTuple);**

**}**

**}**

**/\***

**\* 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 rating\_summerization;**

**import java.io.IOException;**

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

**import org.apache.hadoop.mapreduce.Reducer;**

**/\*\***

**\***

**\* @author Karjout**

**\*/**

**class Summ\_Reducer extends Reducer<Text, MinMaxCountTuple, Text, MinMaxCountTuple> {**

**private MinMaxCountTuple result = new MinMaxCountTuple();**

**@Override**

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

**// Initialize our result**

**result.setAverageRating(0);**

**result.setTotalRating(0);**

**result.setTotalComment(0);**

**int sum = 0;**

**for (MinMaxCountTuple val : values) {**

**if (result.getAverageRating()== 0 || val.getAverageRating() < result.getAverageRating()) {**

**result.setAverageRating(val.getAverageRating());**

**}**

**if (result.getTotalRating()== 0**

**|| val.getTotalRating() > (result.getTotalRating())) {**

**result.setTotalRating(val.getTotalRating());**

**}**

**sum += val.getTotalComment();**

**}**

**result.setTotalComment(sum);**

**context.write(key, result);**

**}**

**}**

**Moving Rating Average by Video\_ID**

**/\***

**\* 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 averagerating\_youtube;**

**import java.io.DataInput;**

**import java.io.DataOutput;**

**import java.io.IOException;**

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

**/\*\***

**\***

**\* @author Karjout**

**\*/**

**public class AverageRating\_CommentCountTuple implements Writable {**

**private int comment\_count = 0;**

**private double video\_rating = 0;**

**public int getComment\_count() {**

**return comment\_count;**

**}**

**public void setComment\_count(int comment\_count) {**

**this.comment\_count = comment\_count;**

**}**

**public double getVideo\_rating() {**

**return video\_rating;**

**}**

**public void setVideo\_rating(double video\_rating) {**

**this.video\_rating = video\_rating;**

**}**

**@Override**

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

**d.writeInt(comment\_count);**

**d.writeDouble(video\_rating);**

**}**

**@Override**

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

**comment\_count = di.readInt();**

**video\_rating = di.readDouble();**

**}**

**@Override**

**public String toString() {**

**return Integer.toString(comment\_count) + " " + Double.toString(video\_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 averagerating\_youtube;**

**import org.apache.hadoop.conf.Configuration;**

**import org.apache.hadoop.conf.Configured;**

**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.util.Tool;**

**import org.apache.hadoop.util.ToolRunner;**

**/\*\***

**\***

**\* @author Karjout**

**\*/**

**public class AverageRating\_Youtube extends Configured implements Tool {**

**/\*\***

**\* @param args the command line arguments**

**\*/**

**@Override**

**public int run(String[] args) throws Exception {**

**Job job = new Job(getConf());**

**job.setJobName("AverageRating\_Youtube");**

**job.setJarByClass(AverageRating\_Youtube.class);**

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

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

**job.setMapperClass(AvgRating\_CommCountMapper.class);**

**job.setMapOutputKeyClass(Text.class);**

**job.setMapOutputValueClass(AverageRating\_CommentCountTuple.class);**

**job.setCombinerClass(AvgRating\_CommCountCombiner.class);**

**job.setReducerClass(AvgRating\_CommCountReducer.class);**

**job.setOutputKeyClass(Text.class);**

**job.setOutputValueClass(AverageRating\_CommentCountTuple.class);**

**boolean success = job.waitForCompletion(true);**

**return success ? 0 : 1;**

**}**

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

**// TODO code application logic here**

**int exitCode = ToolRunner.run(new Configuration(),**

**new AverageRating\_Youtube(), args);**

**System.exit(exitCode);**

**}**

**}**

**/\***

**\* 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 averagerating\_youtube;**

**import java.io.IOException;**

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

**import org.apache.hadoop.mapreduce.Reducer;**

**/\*\***

**\***

**\* @author Karjout**

**\*/**

**public class AvgRating\_CommCountCombiner extends Reducer<Text, AverageRating\_CommentCountTuple, Text, AverageRating\_CommentCountTuple> {**

**private AverageRating\_CommentCountTuple result = new AverageRating\_CommentCountTuple();**

**protected void reduce(Text key, Iterable<AverageRating\_CommentCountTuple> values, Reducer.Context context) throws IOException, InterruptedException {**

**float sum = 0;**

**int count = 0;**

**for (AverageRating\_CommentCountTuple val : values) {**

**sum += val.getComment\_count() \* val.getVideo\_rating();**

**count += val.getComment\_count();**

**}**

**result.setVideo\_rating(sum / count);**

**result.setComment\_count(count);**

**context.write(key, result);**

**}**

**}**

**/\***

**\* 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 averagerating\_youtube;**

**import java.io.IOException;**

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

**import org.apache.hadoop.mapreduce.Mapper;**

**/\*\***

**\***

**\* @author Karjout**

**\*/**

**public class AvgRating\_CommCountMapper extends Mapper<Object, Text, Text, AverageRating\_CommentCountTuple> {**

**// Our output key and value Writables**

**private Text video\_name = new Text();**

**private float v\_rate;**

**private AverageRating\_CommentCountTuple outTuple = new AverageRating\_CommentCountTuple();**

**@Override**

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

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

**String videoId = (fields[0]);**

**if (!fields[6].isEmpty()) {**

**this.v\_rate = Float.parseFloat(fields[6]);**

**} else {**

**this.v\_rate = 0;**

**}**

**video\_name.set(videoId);**

**outTuple.setComment\_count(1);**

**outTuple.setVideo\_rating(this.v\_rate);**

**context.write(video\_name, outTuple);**

**}**

**}**

**Meilleur Youtuber basé sur les vidéos téléchargées**

**package youtubeuploader;**

**import java.io.IOException;**

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

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

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

**import org.apache.hadoop.mapreduce.Mapper;**

**import org.apache.hadoop.mapreduce.Reducer;**

**import org.apache.hadoop.conf.Configuration;**

**import org.apache.hadoop.mapreduce.Job;**

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

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

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

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

**import org.apache.hadoop.fs.Path;**

**public class Youtubetopuploader {**

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

**private Text uploader = new Text();**

**private final static IntWritable occurance = new IntWritable(1);**

**@Override**

**public void map(LongWritable key, Text value,**

**Context context) throws IOException, InterruptedException {**

**String record = value.toString();**

**String str[] = record.split(",");**

**if (str.length >= 7) {**

**uploader.set(str[1]);**

**}**

**context.write(uploader, occurance);**

**}**

**}**

**public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {**

**@Override**

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

**Context context) throws IOException, InterruptedException {**

**int totaloccurance = 0;**

**for (IntWritable value : values) {**

**totaloccurance += value.get();**

**}**

**context.write(key, new IntWritable(totaloccurance));**

**}**

**}**

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

**Configuration conf1 = new Configuration();**

**@SuppressWarnings("deprecation")**

**Job = new Job(conf1, "myyoutube");**

**job.setJarByClass(Youtubetopuploader.class);**

**job.setMapperClass(Map.class);**

**job.setReducerClass(Reduce.class);**

**job.setOutputKeyClass(Text.class);**

**job.setOutputValueClass(IntWritable.class);**

**job.setInputFormatClass(TextInputFormat.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);**

**}**

**}**

**Top 50 des vidéos YouTube préférées**

**package top\_youtuber;**

**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;**

**import java.io.IOException;**

**import java.util.\*;**

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

**public class Top\_Youtube {**

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

**Configuration conf = new Configuration();**

**Job job = Job.getInstance(conf, "Top50");**

**job.setJarByClass(Top\_Youtube.class);**

**job.setMapperClass(TopNMapper.class);**

**job.setReducerClass(TopNReducer.class);**

**job.setOutputKeyClass(Text.class);**

**job.setOutputValueClass(FloatWritable.class);**

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

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

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

**}**

**public static class TopNMapper**

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

**private FloatWritable video\_rating = new FloatWritable();**

**private Text video\_id = new Text();**

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

**) throws IOException, InterruptedException {**

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

**video\_id = new Text(fields[0]);**

**if (!fields[6].isEmpty()) {**

**video\_rating = new FloatWritable(Float.parseFloat(fields[6]));**

**}**

**context.write(video\_id, video\_rating);**

**}**

**}**

**public static class TopNReducer extends Reducer<Text, FloatWritable, Text, FloatWritable> {**

**private Map<Text, FloatWritable> countMap = new HashMap<>();**

**@Override**

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

**// computes the number of occurrences of a single word**

**float sum = 0.0f;**

**int count = 0;**

**for (FloatWritable val : values) {**

**sum += val.get();**

**count++;**

**}**

**countMap.put(new Text(key), new FloatWritable(sum / count));**

**}**

**@Override**

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

**Map<Text, FloatWritable> sortedMap = sortByValues(countMap);**

**int counter = 0;**

**for (Text key : sortedMap.keySet()) {**

**if (counter++ == 50) {**

**break;**

**}**

**context.write(key, sortedMap.get(key));**

**}**

**}**

**}**

**private static <K extends Comparable, V extends Comparable> Map<K, V> sortByValues(Map<K, V> map) {**

**List<Map.Entry<K, V>> entries = new LinkedList<Map.Entry<K, V>>(map.entrySet());**

**Collections.sort(entries, new Comparator<Map.Entry<K, V>>() {**

**@Override**

**public int compare(Map.Entry<K, V> o1, Map.Entry<K, V> o2) {**

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

**}**

**});**

**//LinkedHashMap will keep the keys in the order they are inserted**

**//which is currently sorted on natural ordering**

**Map<K, V> sortedMap = new LinkedHashMap<K, V>();**

**for (Map.Entry<K, V> entry : entries) {**

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

**}**

**return sortedMap;**

**}**

**}**

**Nombre total de vidéos YouTube par catégorie**

**/\***

**\* 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 top10\_categories;**

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

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

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

**/\*\***

**\***

**\* @author Karjout**

**\*/**

**class SortKeyComparator extends WritableComparator {**

**protected SortKeyComparator() {**

**super(IntWritable.class, true);**

**}**

**@Override**

**public int compare(WritableComparable a, WritableComparable b) {**

**IntWritable key1 = (IntWritable) a;**

**IntWritable key2 = (IntWritable) b;**

**int result = key1.get() < key2.get() ? 1 : key1.get() == key2.get() ? 0 : -1;**

**return result;**

**}**

**}**

**/\***

**\* 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 top10\_categories;**

**import java.io.IOException;**

**import org.apache.hadoop.conf.Configuration;**

**import org.apache.hadoop.fs.FSDataInputStream;**

**import org.apache.hadoop.fs.FSDataOutputStream;**

**import org.apache.hadoop.fs.FileStatus;**

**import org.apache.hadoop.fs.FileSystem;**

**import org.apache.hadoop.fs.Path;**

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

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

**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 Karjout**

**\*/**

**public class Top10\_Categories {**

**public static class Map1 extends Mapper<Object, Text, Text, FloatWritable> {**

**private FloatWritable video\_rating = new FloatWritable();**

**private Text video\_id = new Text();**

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

**) throws IOException, InterruptedException {**

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

**video\_id = new Text(fields[0]);**

**if (!fields[6].isEmpty()) {**

**video\_rating = new FloatWritable(Float.parseFloat(fields[6]));**

**}**

**context.write(video\_id, video\_rating);**

**}**

**}**

**public static class Reduce1 extends Reducer<Text, FloatWritable, Text, FloatWritable> {**

**private FloatWritable result = new FloatWritable();**

**@Override**

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

**throws IOException, InterruptedException {**

**int count = 0;**

**float sum = 0, avg = 0;**

**for (FloatWritable val : values) {**

**sum += val.get();**

**++count;**

**}**

**avg = sum / count;**

**result.set(avg);**

**context.write(key, result);**

**}**

**}**

**public static class Map2 extends Mapper<Object, Text, FloatWritable, Text> {**

**@Override**

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

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

**Text video\_id = new Text(row[0]);**

**String rating = row[1];**

**try {**

**FloatWritable ratingg = new FloatWritable(Float.parseFloat(rating));**

**context.write(ratingg, video\_id);**

**} catch (Exception e) {**

**e.printStackTrace();**

**}**

**}**

**}**

**public static class Reduce2 extends Reducer<FloatWritable, Text, Text, FloatWritable> {**

**private static int count = 25;**

**@Override**

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

**for (Text val : values) {**

**if (count > 0) {**

**context.write(val, key);**

**--count;**

**} else {**

**break;**

**}**

**}**

**}**

**}**

**/\*\***

**\* @param args the command line arguments**

**\*/**

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

**Configuration conf1 = new Configuration();**

**Configuration conf = new Configuration();**

**Path inputDir = new Path(args[0]);**

**Path hdfsFile = new Path(args[1]);**

**FileSystem hdfs = FileSystem.get(conf);**

**FileSystem local=FileSystem.getLocal(conf);**

**try {**

**FileStatus[] inputFiles = local.listStatus(inputDir);**

**FSDataOutputStream out = hdfs.create(hdfsFile);**

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

**System.out.println(inputFiles[i].getPath().getName());**

**FSDataInputStream in = local.open(inputFiles[i].getPath());**

**byte[] buffer = new byte[256];**

**int bytesRead = 0;**

**while ((bytesRead = in.read(buffer)) > 0) {**

**out.write(buffer, 0, bytesRead);**

**}**

**in.close();**

**}**

**out.close();**

**} catch (IOException e) {**

**e.printStackTrace();**

**}**

**Job job1 = Job.getInstance(conf1, "Chaining");**

**job1.setJarByClass(Top10\_Categories.class);**

**job1.setMapperClass(Map1.class);**

**job1.setMapOutputKeyClass(Text.class);**

**job1.setMapOutputValueClass(FloatWritable.class);**

**job1.setReducerClass(Reduce1.class);**

**job1.setOutputKeyClass(Text.class);**

**job1.setOutputValueClass(DoubleWritable.class);**

**job1.setCombinerClass(Reduce1.class);**

**FileInputFormat.addInputPath(job1, hdfsFile);**

**FileOutputFormat.setOutputPath(job1, new Path(args[2]));**

**boolean complete = job1.waitForCompletion(true);**

**Configuration conf2 = new Configuration();**

**Job job2 = Job.getInstance(conf2, "Chaining");**

**if (complete) {**

**job2.setJarByClass(Top10\_Categories.class);**

**job2.setMapperClass(Map2.class);**

**job2.setMapOutputKeyClass(FloatWritable.class);**

**job2.setMapOutputValueClass(Text.class);**

**job2.setReducerClass(Reduce2.class);**

**job2.setOutputKeyClass(Text.class);**

**job2.setOutputValueClass(FloatWritable.class);**

**job2.setSortComparatorClass(SortKeyComparator.class);**

**job2.setNumReduceTasks(1);**

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

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

**System.exit(job2.waitForCompletion(true) ? 0 : 1);**

**}**

**}**

**}**

**par catégories**

**/\***

**\* 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 binningbycategories;**

**import java.io.IOException;**

**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.TextInputFormat;**

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

**import org.apache.hadoop.mapreduce.lib.output.MultipleOutputs;**

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

**/\*\***

**\***

**\* @author Karjout**

**\*/**

**public class BinningbyCategories {**

**public static class YouTubeBinMapper extends Mapper<Object, Text, Text, NullWritable> {**

**private MultipleOutputs<Text, NullWritable> mos = null;**

**@Override**

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

**mos = new MultipleOutputs<Text, NullWritable>(context);**

**}**

**@Override**

**protected void map(Object key, Text value, Mapper.Context context)**

**throws IOException, InterruptedException {**

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

**Text Name = new Text(input[3]);**

**String line = Name.toString();**

**if (line.contains("UNA ")) {**

**mos.write("bins", value, NullWritable.get(), "UNA");**

**} else if (line.contains("Autos & Vehicles")) {**

**mos.write("bins", value, NullWritable.get(), "Autos & Vehicles");**

**} else if (line.contains("Comedy")) {**

**mos.write("bins", value, NullWritable.get(), "Comedy");**

**} else if (line.contains("Entertainment")) {**

**mos.write("bins", value, NullWritable.get(), "Entertainment");**

**} else if (line.contains("Film & Animation")) {**

**mos.write("bins", value, NullWritable.get(), "Film & Animation");**

**} else if (line.contains("Gadgets & Games")) {**

**mos.write("bins", value, NullWritable.get(), "Gadgets & Games");**

**} else if (line.contains("Howto & DIY")) {**

**mos.write("bins", value, NullWritable.get(), "Howto & DIY");**

**} else if (line.contains("Music")) {**

**mos.write("bins", value, NullWritable.get(), "Music");**

**} else if (line.contains("News & Politics")) {**

**mos.write("bins", value, NullWritable.get(), "News & Politics");**

**} else if (line.contains("People & Blogs")) {**

**mos.write("bins", value, NullWritable.get(), "People & Blogs");**

**} else if (line.contains("Pets & Animals")) {**

**mos.write("bins", value, NullWritable.get(), "Pets & Animals");**

**} else if (line.contains("Sports")) {**

**mos.write("bins", value, NullWritable.get(), "Sports");**

**} else if (line.contains("Travel & Places")) {**

**mos.write("bins", value, NullWritable.get(), "Travel & Places");**

**} else {**

**mos.write("bins", value, NullWritable.get(), "UnCatogrized");**

**}**

**}**

**@Override**

**protected void cleanup(Mapper.Context context)**

**throws IOException, InterruptedException {**

**mos.close();**

**}**

**}**

**/\*\***

**\* @param args the command line arguments**

**\* @throws java.lang.Exception**

**\*/**

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

**Configuration conf = new Configuration();**

**Job job = new Job(conf, "Binning");**

**job.setJarByClass(BinningbyCategories.class);**

**job.setMapperClass(YouTubeBinMapper.class);**

**job.setNumReduceTasks(0);**

**TextInputFormat.setInputPaths(job, new Path(args[0]));**

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

**// Configure the MultipleOutputs by adding an output called "bins"**

**// With the proper output format and mapper key/value pairs**

**MultipleOutputs.addNamedOutput(job, "bins", TextOutputFormat.class,**

**Text.class, NullWritable.class);**

**// Enable the counters for the job**

**// If there is a significant number of different named outputs, this**

**// should be disabled**

**MultipleOutputs.setCountersEnabled(job, true);**

**System.exit(job.waitForCompletion(true) ? 0 : 2);**

**}**

**}**

**Résultat de chaînage sur Binning pour obtenir les 25 meilleures vidéos par catégorie**

**package top\_youtuber;**

**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;**

**import java.io.IOException;**

**import java.util.\*;**

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

**public class Top\_Youtube {**

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

**Configuration conf = new Configuration();**

**Job job = Job.getInstance(conf, "Top50");**

**job.setJarByClass(Top\_Youtube.class);**

**job.setMapperClass(TopNMapper.class);**

**job.setReducerClass(TopNReducer.class);**

**job.setOutputKeyClass(Text.class);**

**job.setOutputValueClass(FloatWritable.class);**

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

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

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

**}**

**public static class TopNMapper**

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

**private FloatWritable video\_rating = new FloatWritable();**

**private Text video\_id = new Text();**

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

**) throws IOException, InterruptedException {**

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

**video\_id = new Text(fields[0]);**

**if (!fields[6].isEmpty()) {**

**video\_rating = new FloatWritable(Float.parseFloat(fields[6]));**

**}**

**context.write(video\_id, video\_rating);**

**}**

**}**

**public static class TopNReducer extends Reducer<Text, FloatWritable, Text, FloatWritable> {**

**private Map<Text, FloatWritable> countMap = new HashMap<>();**

**@Override**

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

**// computes the number of occurrences of a single word**

**float sum = 0.0f;**

**int count = 0;**

**for (FloatWritable val : values) {**

**sum += val.get();**

**count++;**

**}**

**countMap.put(new Text(key), new FloatWritable(sum / count));**

**}**

**@Override**

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

**Map<Text, FloatWritable> sortedMap = sortByValues(countMap);**

**int counter = 0;**

**for (Text key : sortedMap.keySet()) {**

**if (counter++ == 50) {**

**break;**

**}**

**context.write(key, sortedMap.get(key));**

**}**

**}**

**}**

**private static <K extends Comparable, V extends Comparable> Map<K, V> sortByValues(Map<K, V> map) {**

**List<Map.Entry<K, V>> entries = new LinkedList<Map.Entry<K, V>>(map.entrySet());**

**Collections.sort(entries, new Comparator<Map.Entry<K, V>>() {**

**@Override**

**public int compare(Map.Entry<K, V> o1, Map.Entry<K, V> o2) {**

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

**}**

**});**

**//LinkedHashMap will keep the keys in the order they are inserted**

**//which is currently sorted on natural ordering**

**Map<K, V> sortedMap = new LinkedHashMap<K, V>();**

**for (Map.Entry<K, V> entry : entries) {**

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

**}**

**return sortedMap;**

**}**

**}**

**Recommandation d’abonnés basée sur des abonnés connectés**

**package com.neu.edu.followerreco;**

**import java.io.IOException;**

**import java.util.ArrayList;**

**import java.util.Arrays;**

**import java.util.Comparator;**

**import java.util.HashMap;**

**import java.util.Map.Entry;**

**import java.util.TreeMap;**

**import org.apache.hadoop.conf.Configuration;**

**import org.apache.hadoop.conf.Configured;**

**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.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;**

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

**import org.apache.hadoop.util.Tool;**

**import org.apache.hadoop.util.ToolRunner;**

**public class NewFollowers extends Configured implements Tool {**

**@SuppressWarnings("deprecation")**

**@Override**

**public int run(String[] args) throws Exception {**

**System.out.println(Arrays.toString(args));**

**Job job = new Job(getConf(), "NewFollowers");**

**job.setJarByClass(NewFollowers.class);**

**job.setOutputKeyClass(Text.class);**

**job.setOutputValueClass(Text.class);**

**job.setMapperClass(Map.class);**

**job.setReducerClass(Reduce.class);**

**job.setInputFormatClass(TextInputFormat.class);**

**job.setOutputFormatClass(TextOutputFormat.class);**

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

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

**job.waitForCompletion(true);**

**return 0;**

**}**

**/\*\***

**\* @param args**

**\* @throws Exception**

**\*/**

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

**System.out.println(Arrays.toString(args));**

**int res = ToolRunner.run(new Configuration(), new NewFollowers(), args);**

**System.exit(res);**

**}**

**}**

**class Map extends Mapper<LongWritable, Text, Text, Text> {**

**private Text user = new Text();**

**private Text user1 = new Text();**

**private Text user2 = new Text();**

**@Override**

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

**throws IOException, InterruptedException {**

**int i, j;**

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

**String val[];**

**if (s.length > 1) {**

**val = s[1].split(",");**

**} else {**

**val = null;**

**}**

**user.set(s[0]);**

**if (val != null) {**

**for (i = 0; i < val.length - 1; i++) {**

**user2.set(val[i] + "#-3000");**

**context.write(user, user2);**

**for (j = i + 1; j < val.length; j++) {**

**user1.set(val[i]);**

**user2.set(val[j] + "#1");**

**context.write(user1, user2);**

**user1.set(val[j]);**

**user2.set(val[i] + "#1");**

**context.write(user1, user2);**

**}**

**}**

**user2.set(val[i] + "#-3000");**

**context.write(user, user2);**

**}**

**}**

**}**

**class Reduce extends Reducer<Text, Text, IntWritable, Text> {**

**class ValueComparator implements Comparator<String> {**

**HashMap<String, Integer> base;**

**public ValueComparator(HashMap<String, Integer> base) {**

**this.base = base;**

**}**

**// Note: this comparator imposes orderings that are inconsistent with equals.**

**public int compare(String a, String b) {**

**if (base.get(a) >= base.get(b)) {**

**return -1;**

**} else {**

**return 1;**

**} // returning 0 would merge keys**

**}**

**}**

**@Override**

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

**throws IOException, InterruptedException {**

**HashMap<String, Integer> map1 = new HashMap<String, Integer>();**

**ValueComparator cmp = new ValueComparator(map1);**

**TreeMap<String, Integer> map2 = new TreeMap<String, Integer>(cmp);**

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

**String s[] = values.iterator().next().toString().split("#");**

**if (!map1.containsKey(s[0])) {**

**map1.put(s[0], 0);**

**}**

**map1.put(s[0], map1.get(s[0]) + Integer.parseInt(s[1]));**

**}**

**map2.putAll(map1);**

**ArrayList<String> list = new ArrayList<String>();**

**int i = 0;**

**for (Entry<String, Integer> e : map2.entrySet()) {**

**if (i < 10 && e.getValue() > 0) {**

**list.add(e.getKey());**

**}**

**;**

**i++;**

**}**

**String out = list.toString();**

**context.write(new IntWritable(Integer.parseInt(key.toString())), new Text(out));**

**}**

**}**

**Video based Recommendation using Mahout.**

**ragraw26@ubuntu:/usr/local/lib/mahout/bin$ mahout recommenditembased -s SIMILARITY\_LOGLIKELIHOOD -i /path/to/input/file -o /path/to/desired/output --numRecommendations 25**

**Total Views based on Video ID**

**package youtubeviews;**

**import java.io.IOException;**

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

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

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

**import org.apache.hadoop.mapreduce.Mapper;**

**import org.apache.hadoop.mapreduce.Reducer;**

**import org.apache.hadoop.conf.Configuration;**

**import org.apache.hadoop.mapreduce.Job;**

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

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

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

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

**import org.apache.hadoop.fs.Path;**

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

**public class Topviewed {**

**public static class Map extends Mapper<LongWritable, Text, Text, FloatWritable> {**

**private Text video\_name = new Text();**

**private FloatWritable views = new FloatWritable();**

**@Override**

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

**throws IOException, InterruptedException {**

**String line = value.toString();**

**String str[] = line.split(",");**

**if (str.length >= 5) {**

**video\_name.set(str[0]);**

**float temp = Float.parseFloat(str[5]); //typecasting string to Integer**

**views.set(temp);**

**}**

**context.write(video\_name, views);**

**}**

**}**

**public static class Reduce extends Reducer<Text, FloatWritable, Text, FloatWritable> {**

**@Override**

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

**throws IOException, InterruptedException {**

**int sum = 0;**

**for (FloatWritable val : values) {**

**sum += val.get();**

**}**

**context.write(key, new FloatWritable(sum));**

**}**

**}**

**@SuppressWarnings("deprecation")**

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

**Configuration conf = new Configuration();**

**Job job = new Job(conf, "Top Videos");**

**job.setJarByClass(Topviewed.class);**

**job.setMapOutputKeyClass(Text.class);**

**job.setMapOutputValueClass(FloatWritable.class);**

**job.setOutputKeyClass(Text.class);**

**job.setOutputValueClass(FloatWritable.class);**

**job.setMapperClass(Map.class);**

**job.setReducerClass(Reduce.class);**

**job.setInputFormatClass(TextInputFormat.class);**

**job.setOutputFormatClass(TextOutputFormat.class);**

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

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

**job.waitForCompletion(true);**

**}**

**}**

**Pig Analysis:**

infiles = load 'hdfs://localhost:9000//user/ragraw26/FinalProject/Input/youtube\_data.csv' using PigStorage(',') as

(videoid:chararray,uploader:chararray,age:int,category:chararray,length:int,views:int,rate:int,rating:int,comments:int,related\_id:chararray);

files = FILTER infiles BY category is not null;

grpn\_for\_catagories = group files by category;

cnt\_for\_catagories = foreach grpn\_for\_catagories generate group, COUNT(files.videoid) as counting;

sorted\_for\_catagories\_desc = order cnt\_for\_catagories by counting desc;

top5\_for\_catagories = limit sorted\_for\_catagories\_desc 5;

STORE top5\_for\_catagories INTO 'hdfs://localhost:9000//user/ragraw26/FinalProject/Top5Catagories' using PigStorage(',');

order\_rated\_video = order files by rating desc;

top10\_rated\_video = limit order\_rated\_video 10;

final\_top10\_rated\_video = foreach top10\_rated\_video generate $0,$3,$7;

STORE final\_top10\_rated\_video INTO 'hdfs://localhost:9000//user/ragraw26/FinalProject/Top10Rated' using PigStorage(',');

order\_viewed\_video = order files by views desc;

top10\_viewed\_video = limit order\_viewed\_video 10;

final\_top10\_viewed\_video = foreach top10\_viewed\_video generate $0,$3,$5;

STORE final\_top10\_viewed\_video INTO 'hdfs://localhost:9000//user/ragraw26/FinalProject/Top10Viewed' using PigStorage(',');

top10\_rated\_catagories = foreach grpn\_for\_catagories{

sorted = order files by rating desc;

top10 = limit sorted 10;

generate flatten(top10);

};

top10\_rated\_by\_catagories = foreach top10\_rated\_catagories generate $0,$3,$7;

STORE top10\_rated\_by\_catagories INTO 'hdfs://localhost:9000//user/ragraw26/FinalProject/Top10RatedByCatagories' using PigStorage(',');

top10\_viewed\_catagories = foreach grpn\_for\_catagories{

sorted = order files by views desc;

top10 = limit sorted 10;

generate flatten(top10);

};

top10\_viewed\_by\_catagories = foreach top10\_viewed\_catagories generate $0,$3,$5;

STORE top10\_viewed\_by\_catagories INTO 'hdfs://localhost:9000//user/ragraw26/FinalProject/Top10ViewedByCatagories' using PigStorage(',');

**Analysis and Tableau Dashboard:**



















