***DISCLAIMER**: Submitted to Dublin City University, School of Computing for module CA675: Cloud Technology, 2015/2016. I hereby certify that the work presented and the material contained herein is our own except where explicitly stated references to other material are made.*

Seán O'Sullivan 15211173
Ye Qiu 15210540
SiyiKong 15211374
Mingyao Chen 15210749

Song Recommend Website

Source code available at: https://github.com/osulls59/SongRecommendation

Motivation and Introduction

With the advent of mobile internet, the way people listen to music has changed dramatically in recent years. Previously people would rely on purchasing vinyl records, cassette tapes or CD's. This would limit the person to listening only to the music in his or her collection. However, in recent years, the internet is being increasingly used as a platform for listening to music. Gone are the days where you can only listen to one of the six CD's allotted in the player in your car boot. Nowadays iTunes, Soundcloud, Spotify or Google Play Music provide people access to millions of songs from a plethora of artists without the need for them to carry or even own a copy of the music they listen to.
In a similar manner to the Netflix Prize, awarded in September of 2009, we aim to create a recommendation algorithm that will instead focus on providing *song* recommendations based on the user's input.

Environment

The related tools we have used are eclipse IDE, tomcat 7.0, mapreduce api and DFS location
server. We considered that the data files have to be distributed in the DFS server. So the first
step is to make Hadoop mode distributed. Then we installed Hadoop-Eclipse-Plugin to compile and run the MapReduce project on eclipse and we added a new Hadoop location with some settings. After that, we created the Mapreduce project on eclipse.

Data Pre-preprocessing

Data Description

The data used for the application was the Million Song Dataset. This was obtained from the Laboratory for the recognition and organisation of speech and Audio. (Million Song Dataset, official website by Thierry Bertin-Mahieux, available at: http://labrosa.ee.columbia.edu/millionsong/)

The website makes available a comprehensive description of over 1,000,000 songs with fields for each song such as the name of the song, the artist, the genre of music for which the song falls under, the tempo of the track to name but a few (a full description of each of the fields is explained below).

The high level of detail the dataset includes means that the entire dataset of 1,000,000 songs is around 280GB. With our limited computing resources it was not feasible, even while utilizing the Mapreduce paradigm, to process the entire dataset in real time so for this reason we opted to use a subset of the Million Song Dataset which includes a random selection of 10,000 songs. Using only this subset, the data came to a much more manageable 2.2GB. It should be noted that given access to more computing resources it would be trivial to switch our application to use the full 280GB dataset.

Field name	Type	Description
analysis sample rate	float	sample rate of the audio used
artist 7digitalid	int	ID from 7digital.com or -1
artist familiarity	float	algorithmic estimation
artist hotttnesss	float	algorithmic estimation
artist id	string	Echo Nest ID
artist latitude	float	latitude
artist location	string	location name
artist longitude	float	longitude
artist mbid	string	ID from musicbrainz.org
artist mbtags	array string	tags from musicbrainz.org
artist mbtags count	array int	tag counts for musicbrainz tags
artist name	string	artist name
artist playmeid	int	ID from playme.com, or -1
artist terms	array string	Echo Nest tags
artist terms freq	array float	Echo Nest tags freqs
artist terms weight	array float	Echo Nest tags weight
audio md5	string	audio hash code
bars confidence	array float	confidence measure
bars start	array float	beginning of bars, usually on a beat
beats confidence	array float	confidence measure

beats start	array float	result of beat tracking
danceability	float	algorithmic estimation
duration	float	in seconds
end of fade in	float	seconds at the beginning of the song
energy	float	energy from listener point of view
key	int	key the song is in
key confidence	float	confidence measure
loudness	float	overall loudness in dB
mode	int	major or minor
mode confidence	float	confidence measure
release	string	album name
release 7digitalid	int	ID from 7digital.com or -1
sections confidence	array float	confidence measure
sections start	array float	largest grouping in a song, e.g. verse
segments confidence	array float	confidence measure
segments loudness max	array float	max dB value
segments loudness max time	array float	time of max dB value, i.e. end of attack
segments loudness max start	array float	dB value at onset
segments pitches	2D array float	chroma feature, one value per note
segments start	array float	musical events, ~ note onsets
segments timbre	2D array float	texture features (MFCC+PCA-like)
similar artists	array string	Echo Nest artist IDs (sim. algo. unpublished)
song hotttnesss	float	algorithmic estimation
song id	string	Echo Nest song ID
start of fade out	float	time in sec
tatums confidence	array float	confidence measure
tatums start	array float	smallest rythmic element
tempo	float	estimated tempo in BPM
time signature	int	estimate of number of beats per bar, e.g. 4
time signature confidence	float	confidence measure
title	string	song title
track id	string	Echo Nest track ID
track 7digitalid	int	ID from 7digital.com or -1
year	int	song release year from MusicBrainz or 0

Table taken from: http://labrosa.ee.columbia.edu/millionsong/pages/field-list

Data Cleaning

When the data is downloaded, the data is arranged in folders A to Z in the HDF5 format, with each song contained in it's own .h5 file. Python code was obtained from github (https://github.com/rcrdclub/mm-songs-db-tools) that made it possible to convert this into a much more user/map reduce friendly csv form. The script recursively converted each .h5 file into a line in the user specified csv file. While this was a convenient solution, it did lead to one unforeseen issue---the delimitation of the file. By default, the conversion code output the csv as comma delimited. However, some of the fields such as the song's genre's (labelled above as *artist mbtags*) consisted of an array---which in turn was comma delimited(","). Hence, the conversion code was altered to delimit each field in the dataset with a semicolon (";"). This enabled the map reduce code to differentiate new fields from arrays within a record.

Recommendation

The general recommendation process of this program is shown as below:

- Get user input song title.
- Get the record of the song based on the title.
- Mapper program: prepare the features needed in following similarity calculate step.
- Reducer program: Calculate the similarity between user selected song and each song stored in HDFS, and find ten max values of *Pearson product-moment correlation coefficient.*

Name Check

Since the whole *Million Song Dataset* could be extremely huge, we only download part of the data (2GB). Then users can not find all the songs they want in our database. Thus we provide a name check function: Once user type in part of the title, the system will automatically check the word and query songs which include the word. This function contain two parts: Ajax as well as back-end MapReduce Program. The back-end use regex expression to match titles which contain the word.

Ajax:

```
function titleCheck() {
  $("#PosiibleSongs").hide();
  $("#10songs").hide();
  $.ajax({
       type:'post',
       url:'GuestbookServlet',
       dataType:'json',
       data:{word:''+ $("#textinputbox").val()} ,
       success:function(jsonob){
         $("#PosiibleSongs").html("<h4 style='color:white;'>Do you means:</h4>");
         var songs = jsonob.title;
         if(songs.length != 0) {
           var i = 0;
           for(; i < songs.length; i ++) {
             $("#PosiibleSongs").append("<h5 style='color: white'> " + songs[i] + " </h5>")
           }
           $("#PosiibleSongs").show();
         }
      }
     });
```

Figure 1.1 name check code

MapReduce:

```
public static class SongMapper extends Mapper<LongWritable, Text, Text,
Text> {
     @Override
     protected void map(LongWritable key, Text value, Context context) throws IOException,
      InterruptedException {
      if (key.get() > 0) {
/          String[] lines = new CSVParser().parseLine(value.toString());
       String[] lines = new CSVParser(';').parseLine(value.toString());
           String regex = ".*(?i)"+NameCheck123.java+".*";
           Pattern pattern = Pattern.compile(regex);
           Matcher matcher = pattern.matcher(lines[11]);
           boolean rs = matcher.find();
           Matcher matcher1 = pattern.matcher(lines[50]);
           boolean rs1 = matcher1.find();
             if(rs){
              // context.write(new Text(lines[11] + ": " + lines[50]));
               context.write(new Text(lines[11]), new Text(lines[50]));
             }

             if(rs1){
               // context.write(new Text(lines[11] + ": " + lines[50]));
              context.write(new Text(lines[11]), new Text(lines[50]));
             }
```

Figure 1.2 name check code in mapreduce

Recommend Algorithm

For Recommendation MapReduce program. The mapper program is used to load data into memory then extract out information we need. Then group data by artist name which allow fast process in Reduce program. Plus, to better the search result, we only take consider songs by similar artists. Corresponding code shown as below:

```
public void map(Text key, Text value, Context context) throws IOException, InterruptedException {
  Text feature = new Text();
  String line = value.toString();
  String[] terms = line.split(",");
  String filtered = terms[46] + "," + terms[17] + "," + terms[20] + "," + terms[28];

  feature.set(filtered);
  key.set(terms[7]); // artist name as the key
  if (similar_artists.contains(terms[7]))
    if(Double.parseDouble(terms[2]) > 0.2 && Double.parseDouble(terms[3]) > 0.2)
      context.write(key, feature);
}
```

Figure 1.3 recommend algorithm code in mapreduce

The recommendation algorithm we use is *Pearson product-moment correlation coefficient* which is implemented using mapreduce to calculate the similarity of two vectors. The corresponding formula is shown below:

$$\rho_{X,Y} = \frac{\operatorname{cov}(X,Y)}{\sigma_X \sigma_Y} = \frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y}$$

Corresponding code shown as below:

```
public double correlation(List<Double> x, List<Double> y) {
  double coefficient = 0.0;

  coefficient = this.cov(x, y) / (this.std(x) * this.std(y));
  return coefficient;
}

/**
```

Figure 1.4 corresponding code

Web Application

User Interface

The web application contains two pages. The first page is a welcome page, once user click on the “ RECOMMEND ME A SONG” button, the website will redirect to ”SONG RECOMMENDA” page.

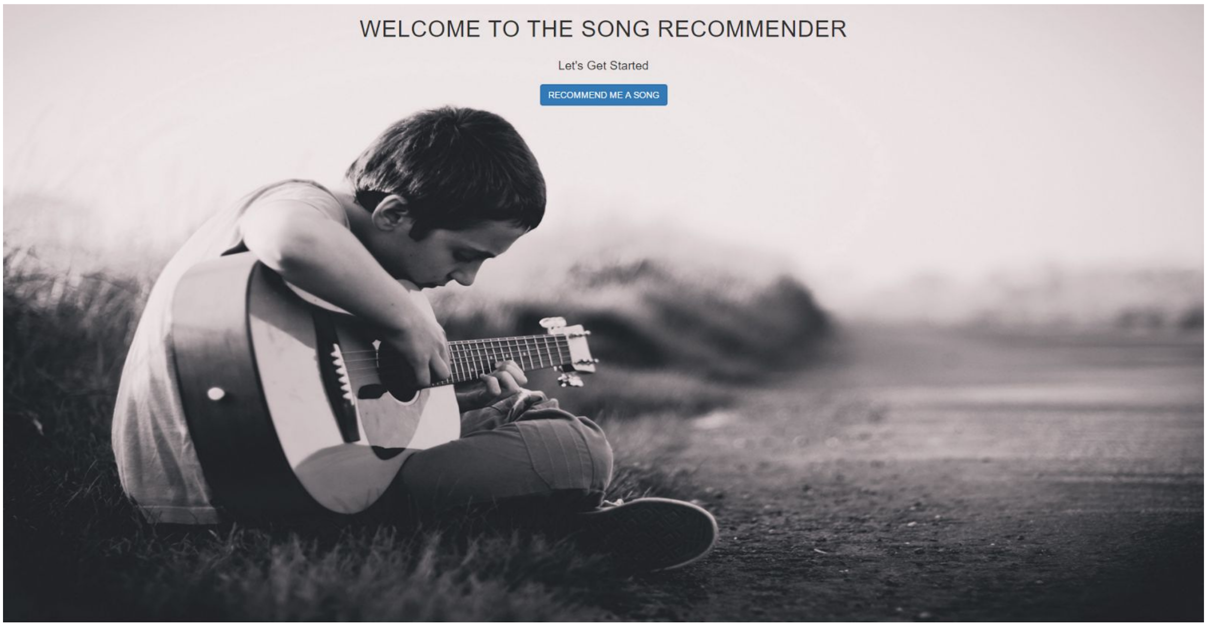

Figure 1.5 Welcome Page

In “SONG RECOMMEND” page, user are allowed to input the title of a song, once user enter part of title, the website will automatically check relative songs exists in our database. Shown in figure 1.3

Figure 1.6 Song Recommend Page

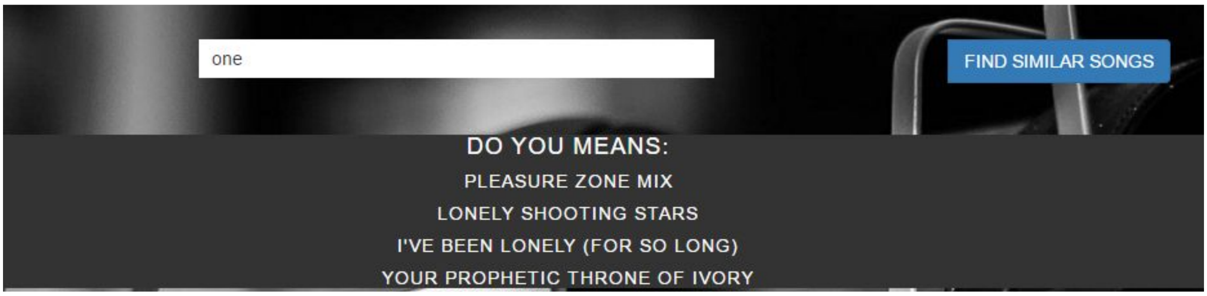

Figure 1.7 Title Check

Once user enter the complete title of the song, the recommended 10 songs will be presented on web interface, shown in Figure 1.4.

Figure 1.8 Similar Songs

Back-End

For the servlet part, it is served as an intermediate layer for the communication between the user interface , the data processing and MapReduce. For the Map/reduce part, we added some configurations to get the mapreduce to work correctly.

```
Configuration conf=new Configuration();

/*configuration of mapreduce */
        conf.set("fs.defaultFS","hdfs://localhost:9000");   //allocate namenode
  conf.set("fs.hdfs.impl", "org.apache.hadoop.hdfs.DistributedFileSystem");
  conf.set("mapreduce.framework.name", "yarn");  // set to use yarn framework
  conf.set("fs.AbstractFileSystem.hdfs.impl", "org.apache.hadoop.fs.Hdfs");
  conf.set("yarn.resourcemanager.scheduler.address", "0.0.0.0:8030");
  conf.set("yarn.resourcemanager.address","0.0.0.0:8032");  //allocate resourcemanager
  conf.set("yarn.scheduler.maximum-allocation-mb", "512");  //set maxium of schedular
      conf.set("mapreduce.map.memory.mb", "256");
```

Figure 1.9 configuration code in mapreduce

In our development environment, we were using the Pseudo-Distributed mode of hadoop. So, we need to specify the namenode address, resourcemanager address and resourcemanager scheduler address. If the application switch to a Fully-Distributed operation environment, the configuration of mapreduce also needs to be changed to match the Fully-Distributed operation environment.