Hadoop LastFM Analysis

CSinParallel Project

August 13, 2014

CONTENTS

This module demonstrates how hadoop and WMR can be used to analyze the lastFM million song dataset. It incorporates several advanced hadoop techniques such as job chaining and multiple input. Students should know how to use the WMR hadoop interface before beginning this module.

The dataset was obtained from Columbia University's LabROSA. However it has been converted into a format that is easier to work with on WMR. The edited dataset is also much smaller since it doesn't include the audio analysis information. If you would like the smaller dataset for your own WMR cluster please contact JLyman@macalester.edu

CHAPTER

ONE

THE MILLION SONG DATASET

Last.fm is a popular music recommendation website that tracks what it's users listen to and then suggests similar songs. It provides an API which can be used to retrieve metadata about specific songs.

Researchers at Columbia's LabROSA used this API to generate a dataset containing 1,000,000 songs from 44,745 artists. For our purposes this data has been split into 3 different tab separated files.

Each file has a different prefix which hadoop passes to the mapper as a key. The rest of the data is passed to the mapper as the value. Each file contains several fields so it is necessary to call value.split('t') to access them.

System-dependent Alert

The path of each of the datasets shown below may not be the same on your WMR system. It is correct for this WMR server:
selkie.macalester.edu/wmr

- **/shared/lastfm/similars.tsv** contains information about artists who are similar to each other. The prefix used to tag each group of artists is "similar" (the key for the mapper). The first value of the data after that tag is the id of an artist and the rest of the values are the ids of similar artists. The list of similar artists varies in size. If artist A's list contains B and C there is no guarantee that B's list contains C or A. The list of similar artists varies in length and may be empty.

  Here is what a small portion of the first line of this file looks like:

  ```
  similar     ARRPV2V1187B9B6312      ARDDLVP1187B98CB8A      AR3M0PY119B86695E3      ARC1X4C1187E
  ```

- **/shared/lastfm/terms.tsv** contains musical terms associated with artists. The prefix beginning each line is "term" (the key for the mapper). The first value of the data is an artist id and the rest of the values are comma separated triplets representing terms associated with the artist. They can be separated by calling term.split(',')

  - The first value is the term itself. It may be a genres like "rock" or "pop" or a descriptor like "london"
  - The second value is the frequency with which that term is used in reference to the artist, it is a float from 0-1
  - The last value is the weight of the term which provides a a measure of how well a given term is to describes the an artist. For example 'rock' is frequently used to describe the Beatles, but "british invasion" is more descriptive so it has a higher weight. The weight is a float from 0-1

  there are a variable number of terms associated with each artist and there may be none. Here is a small portion of the first line of this file:

  ```
  term        ARRPV2V1187B9B6312      hard house,0.934636697674,1.0   viking metal,0.849886186054,
  ```

- **/shared/lastfm/metadata.tsv** is prefixed with “metadata”. The data contains 25 different fields of metadata about a given song, which are explained in the following chart. These fields may be null (for strings) on nan (not a number, for floats).

index	value	Description
0	track id	String
1	title	String
2	release name (album)	String
3	year	Int
4	artist name	String
5	artist id	String
6	artist familiarity	Float 0-1 How well known an artist is
7	artist hotness	Float 0-1 Current popularity of an artist
8	artist latitude	Float
9	artist longitude	Float
10	artist location	String
11	hotness	Float 0-1 current popularity of a song
12	danceablity	Float 0-1
13	duration	Float number of seconds in a song
14	energy	Float 0-1
15	loudness	Float 0-1
16	end of fade in	Float
17	start of fade out	Float
18	tempo	Float tempo in beats per minute
19	time signature	Int number of beats per measure
20	time signature confidence	Float 0-1 confidence in the above number
21	mode	1 for major 0 for minor
22	mode confidence	Float 0-1 confidence in the above number
23	key	Int C=0, C#=1, D=2...
24	key confidence	Float 0-1 confidence in the above number

Here is the first line of this file (intentionally wrapped so you can see all the values; it contains no newlines except for the ending one it the real file):

```
metadata    TRAMMDT128F934E8C5      Wicked City     Stuntrock (Original Soundtrack) 0
Sorcery     ARRPV2V1187B9B6312      0.433976792653  0.335432931443  40.82559        -74.10874
null        nan     0.0     257.14893       0.0     -10.068 0.195   253.063 122.771 4       0.812
0   0.439   4       0.383
```

Note: The year in the above line of data, the third element counting from 0 after the “metadata” tag prefix, is given as 0.

CHAPTER

TWO

FUN WITH KEY SIGNATURES

Let's get our hands dirty by answering a practice question: What is the most common key signature in the dataset?

A key signature is made up by a key (C, G#, etc) and a mode, either major or minor (these aren't the only modes, but they are the only ones in the dataset). Both the key and the mode are important, because A minor and C major contain the same notes so if our mode is incorrect we will get bad results.

Luckily the dataset provides us with a measure of how accurate it's guesses for the key and the mode are. These are both floats from 0-1 so to find the confidence in the key signature we'll multiply them, that way if the key is certain but the mode is totally unknown, the confidence will be low.

2.1 Coding the Hadoop Job

A glance at the chart from last chapter tells us that the key and key confidence are stored at indices 23 and 24 respectively and that the mode and mode confidence are stored at indices 21 and 22 respectively.

Armed with this information we can write a mapper that emits a key signature as a key and the confidence as a value. We'll also perform a basic sanity check on our data by testing to see if all 25 fields are present. It's good practice to sanity check data in the mapper because you can never be certain that your data is pure.

Our `mapper` looks like this:

```
def mapper(key, value):
  data = value.split('\t')
  if len(data) == 25:
    keySig = (data[23], data[21])
    confidence = float(data[24]) * float(data[22])
    Wmr.emit(keySig, confidence)
```

Note: Remember, WMR interprets all keys and values as strings, however we're using a tuple as a key and a float as a value. This is okay since they get automatically cast by WMR, we just have to remember to recast them in the reducer. Python's eval() method is useful for getting tuples from strings

Our `reducer` will sum up all of the confidences. This way songs that have higher confidences will have more influence on the total than songs with uncertain keys. It also turns the key signatures from numbers into something more human readable. Doing the conversion in the reducer instead of the mapper saves a lot of work because, the calculation is only performed once per each of the 24 keys rather than once per each of the million songs

```
def reducer(key, values):
  keys = ['C','C#','D','D#','E','F','F#','G','G#','A','A#','B']
  keySig, mode = eval(key)
  keySig = keys[int(keySig)]
  if mode == '0':
    keySig += 'm'
  count = 0.0
  for value in values:
    count += float(value)
  Wmr.emit(key, count)
```

After running the job we find that the most common key is G major and the least common is D#/E flat minor.

2.2 Going Further

Why is G the most popular key? One reason could be the guitar. The fingerings for chords in the key of G are all very simple so maybe artists pick to G because it's easy. If this theory is true, then genres like rock and country that use more guitars should use the key of G more often than genres like Hip Hop and electronic.

Unfortunately our dataset only has artist level tags, so we will need to create a filtering job that only outputs songs by artists who have been tagged with a specific genre.

This means that our hadoop job will have to read input from both the terms file and the metadata file. We can do this by using /shared/lastfm/ as the input path. Since it is a folder, all of the files in the folder are used as input. We want to pull different pieces of information from each of these files

- **From metadata:** the key signature and confidence of a song
- **From terms:** whether the genre is in the terms list and has a weight greater than 0.5

We want to send all of this information to the reducers sorted by artist. The artist ID of a song is at index 5 of the metadata file and the artist ID is at index 0 in the terms file. We can let the reducer know what information is being passed to it by emitting tuples where the first value is a flag stating what the second value is.

With this information we can write a `mapper (genreMapper.py)`.. Remember to perform the sanity check on the metadata. Unfortunately we can't run the same check on the other files because they have variable line lengths.

```
def mapper(key, value):
  genre = "rock"
  data = value.split('\t')
  if key == "metadata" and len(data) == 25:
    artist = data[5]
    keySig = (data[23], data[21])
    confidence = float(data[24]) * float(data[22])
    Wmr.emit(artist, ("song", (keySig, confidence)))
  elif key == "term":
    artist = data[0]
    for triplet in data[1:]:
      term, freq, weight = triplet.split(',')
      if term == genre and float(weight) > 0.5:
        Wmr.emit(artist, ("term", True))
```

Our `reducer (genreReducer.py)` will need to take all of this data and only emit the songs by artists who are tagged with the the genre.

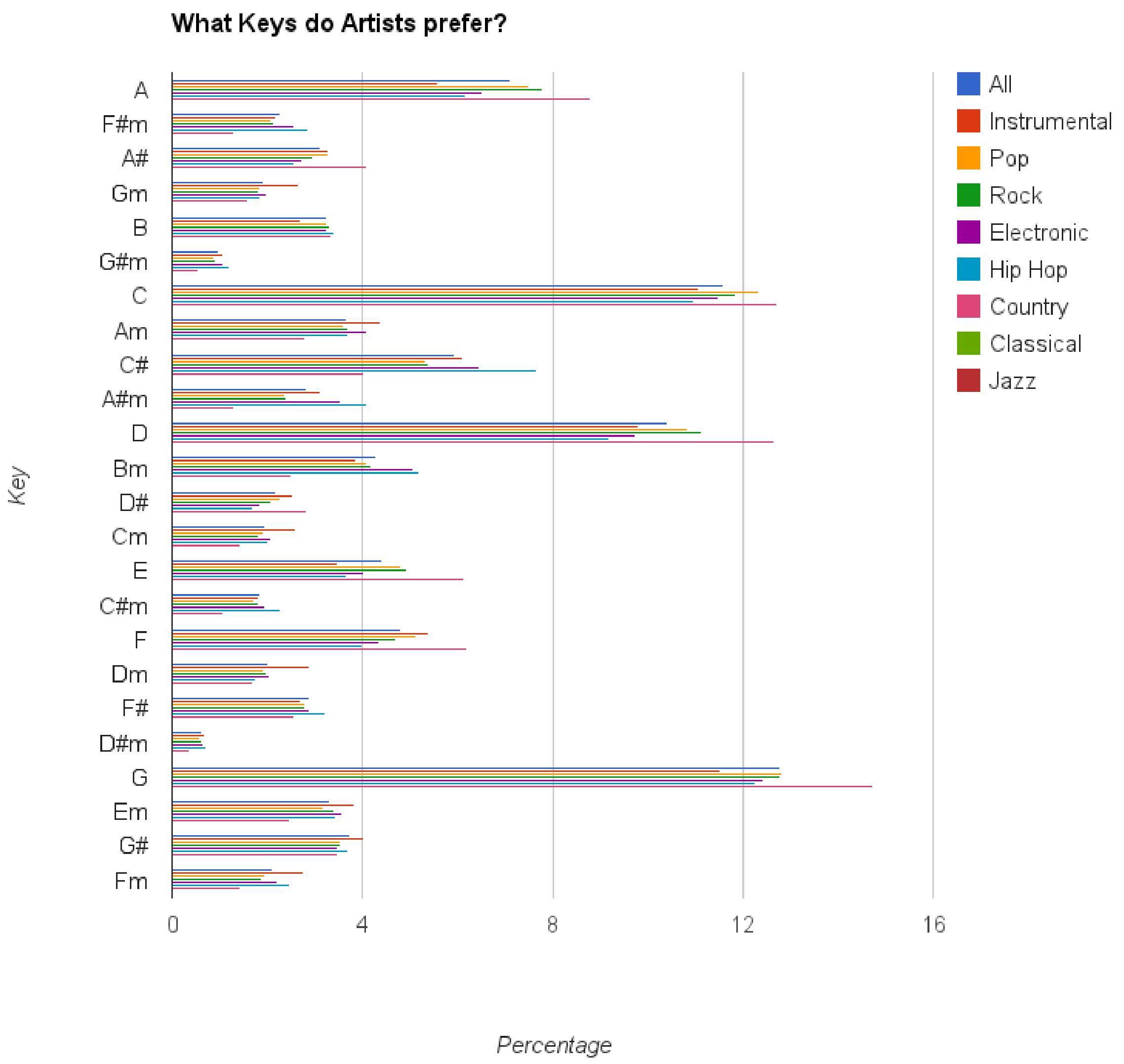
```
def reducer(key, values):
  isMatch = False
  songPairs = []
  for value in values:
    flag, data = eval(value)
    if flag == "term":
      isMatch = data
    elif flag == "song":
      songPairs.append(data)
  if isMatch:
    for keySig, confidence in songPairs:
      Wmr.emit(keySig, confidence)
```

After running this job we are left with a list of key signatures and confidences. We still need to add up the confidences for each of the key signatures. We can do this by passing our list to the `reducer (avgKeyReducer.py)` from the first part of this module. To use the output of a Wmr job as input for another, just click the 'Use Output' on either the top or the bottom of the page.

To pass our data straight to the reducer we'll use what's known as the `identity mapper`:

```
def mapper(key, value):
  Wmr.emit(key, value)
```

Try running this compound job for different values of genre. All of the tags in the terms file are lowercase. Once you've finished make a graph of the percentage of songs that are in each key per genre. It should look something like this:



2.3 Interpreting the results

It looks like G is the most popular key for every genre but classical where it barely looses out to C. In country music G is a heavy favorite along with C and D which are also relatively easy keys to play on the guitar. However G is also very popular in electronic and hip hop, genres where the voice is often the only acoustic instrument.

Overall it seems like the guitar does have some effect on an artist's choice of key, but it can't be explained by guitar tuning alone.

2.4 Challenge

Can you find a way to find the counts for 6 different genres using only one chain of jobs?

CHAPTER

THREE

CHALLENGE PROBLEMS

Now that you're familiar with the dataset try some of these challenge problems. Be careful, some songs are missing fields

- Find the average artist familiarity. This information is stored in the metadata file at index 6 so you'll have to find a way to make sure that each artist is only counted once
- Find the average danceablity of an artist's songs. Compare it with the artist's familiarity. Is there a correlation between them?
- Find the duration of the fade in/out for each song. Plot your results by year. How have tastes changed over time?
- Do similar artists have similar danceabilities, energies, and loudnesses?