Week 8

From the Expert: Implementing a Recommender System using the Hadoop ecosystem

The virtual machine used in the class is the product of Cloudera's Distribution including Apache Hadoop (CDH), which contain a number of tools such as Sqoop, Hive, Mahout, Python, and R. The PATH environment variables contain the command for these tools so there is no need to specify the executable locations, unless noted otherwise.

In this course, you will use VMs for CDH 5.1.x (www.cloudera.com), which is a single-node hadoop cluster.

Exercises

For these exercises you will use the Movielens 1M dataset, which contains over 1M ratings by over 6000 users for almost 4000 movies. The web page and source for this data is http://grouplens.org/datasets/movielens/. GroupLens is a research project in the department of Computer Science and Engineering at the University of Minnesota.

This dataset contains 3 files organized as follows below. You can read the full dataset description from this link: http://files.grouplens.org/datasets/movielens/ml-1m-README.txt

1. *movies.dat* is a list of movies in double-colon (::) separated format movieid::title::genres, with the following 3 fields:
a. A movie id number
b. The movie title and the year it was introduced
c. A list of zero or more movie genres separated by the vertical-pipe character "|"

2. *users.dat* is a list of users with demographic data in double-colon (::) separated format userid::gender::age::occupation::zipcode, with the following 5 fields:
a. A user id number
b. Gender M or F
c. Age range (denoted by the first year of that range).
d. A number reflecting occupation
e. The user's Zip code in 5-digit format.

3. *ratings.dat* is a list of ratings by users in double-colon (::) separated format userid::movieid::rating::timestamp, with the following 4 fields:
a. A user id number
b. A movie id number
c. A rating 1-5
d. A timestamp for when the rating was given

Locate movie lens data on shared folder in Cloudera VM
cd /home/cloudera/Exercises/ml-1m
ls -l

You can briefly examine the data using the UNIX "cat" or "more" or "tail" commands, and use more sophisticated ways to inspect the characteristics of this data. This is actually a relatively small dataset. However, with true Big-data, the data volume will be too large to inspect using such simple techniques. It will need to be placed into the hadoop file system HDFS and run map-reduce tasks for the inspection and "cleaning". We will use this dataset to emulate "Big-data" techniques.

The exercises will cover these 4 steps:

Step 1: Introduction to Hadoop commands and experiments with the Hadoop commands

Step 2: Preprocessing data using Python (e. g. clean up, transform) and create comma separated files to use in later steps

Step 3: Create Hive tables and queries from the files

Step 4: Build the basic recommender system using Mahout

Step 1: Introduction to Hadoop Commands and experiments with the Hadoop commands
The interaction with Hadoop is through a command line wrapper called *hadoop*. Most Hadoop commands are somewhat similar to UNIX commands and structured as:

hadoop fs –[unix-command] –[unix_arguments] [hadoop command arguments]

Typically, The error information is sent to stderr, and the output is sent to stdout.

For example:
Unix: $ ls -l
Hadoop: $ hadoop fs –ls

("hadoop fs" without –l is equivalent with normal unix ls with "–l")

If you type "hadoop fs" by itself you will see a list of possible commands and arguments.

The directory structure in HDFS is completely separate from the structure of the local file system. The command "ls /" shows the contents in the root directory of the local file system, whereas, the command "hadoop fs –ls /" shows the contents in the root directory in the HDFS filesystem; they are completely separate storage namespaces even though HDFS actually uses local storage; you only interact with the contents via the hadoop/hdfs interface.

a) Prepare to store the data into the ***HDFS*** file system by creating directories for it.

mkdir – make/create directories
Example:
$ hadoop fs -mkdir -p /user/data/movielens/raw
$ hadoop fs -mkdir /user/data/movielens/raw/users
$ hadoop fs -mkdir /user/data/movielens/raw/movies
$ hadoop fs -mkdir /user/data/movielens/raw/ratings

The first command with '-p' option is used to make parent directories as needed, and no error if exists.

b) Put the data into HDFS. ***Make sure you are in the /home/cloudera/Exercises/ml-1m*** directory mentioned earlier.
put – copy single or multiple source from local file system to the destination HDFS filesystem
Example:
$ hadoop fs -put users.dat /user/data/movielens/raw/users
$ hadoop fs -put movies.dat /user/data/movielens/raw/movies
$ hadoop fs -put ratings.dat /user/data/movielens/raw/ratings

c) Confirm that the data is there.
ls – return a list of the files in the directory
Example:
$ hadoop fs -ls -R /user/data/movielens

d) Now make a subdirectory to contain the "cleaned" data that we will create via map-reduce from the step 2:
$ hadoop fs -mkdir /user/data/movielens/cleaned

Some other commands:
e) cat – examine the content of the output file (practically, this is not recommend for the big-data files, but can be used in this case)
Example:
$ hadoop fs –cat /user/data/movielens/raw/ratings/ratings.dat

f) tail – display last kB of the file to standard out
Example:
$ hadoop fs –tail /user/data/movielens/raw/users/user.dat

g) get – copy the HDFS file to the local directory
Example
$ hadoop fs –get /user/data/movielens/raw/ratings/ratings.dat ratings_cpy.dat

To see more information about hadoop visit this web site:
http://hadoop.apache.org/docs/r2.4.1/hadoop-project-dist/hadoop-common/FileSystemShell.html

The directories used in the exercises include:

- /home/cloudera/Exercises/ml-1m

directory in the local file system for data sets

- /home/cloudera/Exercises/scripts

directory in the local file system for script files used in the exercises

- /user/data/movielen/raw

directory in HDFS for the raw data (users, movies, ratings as subdirectories)

- /user/data/movielen/cleaned

directory in HDFS for the cleaned data (users, movies, ratings as subdirectories)

Step 2: Preprocessing data using Python

Python is a high-level language that runs on many computer platforms such as Linux, Mac OS X, and Microsoft Windows. Python supports object-oriented programming. In addition, there are many modules including re (regular expression) and sys (system), which make complex tasks quick and easy and can reduce the amount of code.

Python is similar to other script languages such as perl. The important difference is using white spaces rather than curly braces to represent code blocks. Also, the code blocks are set off with a colon character (:). Another difference is Python is dynamically typed. A declared variable can be assigned to different types. We can interact with Python interpreter directly at the shell prompt. It is also common to store the code in a file using .py extension. It is important to make sure that this file is executable using command such as chmod +x test.py.

After adding the following line at the top of the file, the .py file can be run directly from the UNIX shell. This line signifies that Python interpreter is used to execute this script.

```
#! /usr/bin/env  python
```

Since the objective of this step is to practice preprocessing the data, the tasks that will be performed will related to files under the directory /user/data/movielens/raw as follows:

1) For ratings.dat,
   a) convert the file in into a tab separated file as this format:
   userid<tab>movieid<tab>rating<tab>timestamp.
2) For movies.dat,
   a) convert data into a tab separated file
      movieid<tab>title<tab>year<tab>list_of_genres_num
   b) separate year from the title into another field and
   c) transform the genres into category number as format:
      Action|Adventure|Western => 1,2,18

   For instance, transforming genres text label into category number:

   Action 1
   Adventure 2
   …..
   Western 18

A movie can have more than one genre. In this case, convert each genre to its corresponding number. For movie with no associated genres, which appear as blank genres in the movies.dat file will be replaced with number 99.

3) For users.dat, the focus is on fixing the zip code. The zip code format is not consistent and in various forms such as 5 digits, 5 digits '-' 4 digits (xxxxx-xxxx), more than 6 digits, or blank. Here, the new zip code will be converted to only 5 digits format. Therefore, the zip code with 5 digits '-' 4 format will be truncated and used only 5 digits before dash. The format with 6 or more digits will be truncated to 5 digits as well.

Many times, we want to extract valuable information from the log file. However, the delimited character formats such as comma, tab may not be available. The matching values have to rely on the observed patterns. This can be done by regular expressions, which support by *re* module from Python. In this case, the movie title contains the year, for example "Top Gun (1985)". This is often a necessary part of the name label to differentiate a movie that was remade in later years. However, we want to extract it as part of this exercise so that it may be queried.

You might want to try coding your own program first. However, you can see the hint in the given script under directory /home/cloudera/Exercises/scripts. A script is provided that will invoke mapReduce to clean data for each of the 3 data files: do_mapreduce.sh. This uses the "streaming" function of mapReduce, which allows any scripts to be used with mapReduce. In this case, we use the python language to do the cleaning. The script that invokes the mapReduce is written in bash. It is provided for convenience since there will be a lot to type! It is a good idea to look inside the script to see what it is doing to invoke hadoop.

If you want to see how the scripts work, type the following commands:
cd /home/cloudera/Exercises/scripts
./do_mapreduce.sh users (this invokes the clean_users_dat.py python script)
./do_mapreduce.sh movies (this invokes the clean_movies_dat.py python script)
./do_mapreduce.sh ratings (this invokes the clean_ratings_dat.py python script)
The above commands will generate output data in /user/data/movielens/cleaned. Once generated, it is time to load the data into hive in step 3.

The do_mapreduce.sh script invokes the python script to be executed using hadoop streaming such as:

```
$ hadoop jar $STREAMJAR \
  -Dmapred.reduce.tasks=0 \
  -input /user/data/movielen/raw/ratings \
  -output /user/data/movielen/cleaned/ratings \
  -mapper clean_ratings_dat.py \
  -file clean_ratings_dat.py
```

Hadoop streaming is a utility that allows you to run Map/Reduce jobs with script or any executable as the mapper and reducer. Note that, the backslash ("\") at the end of the command line indicates that the command line is not completed, and it continues to the

next line. However, if you prefer one single line, you should type the command without backslash.

The explanations are given here:
jar runs a jar file.
-Dmapred.reduce.tasks=0
This means that input data will be processed using a map function only. Therefore, Dmapred.reduce.tasks is set to 0. The outputs of the mapper will be the final output of the job.
-input is the path of the input data directory
-output is the path of the output data directory
-mapper specify the file name for mapper executable
-file make copy of file available to the tasks

Step 3: Create Hive tables

The purpose of this step is to create Hive tables so that it is easy to do queries for filtering, combining, and analyzing the data.

Hive shell allows you to run the query interactively. HiveQL is similar to SQL.

Change the directory using:
cd /home/cloudera/Exercises/scripts

Then, type these commands:
$ hive -f create_users_table.hql
$ hive -f create_movies_table.hql
$ hive -f create_ratings_table.hql

Using –f option tells Hive to execute the HiveQL code in the referred file (SQL from files). Each hive script provided does 3 things: 1) Drop the table if it exists, 2) create the new table, 3) load the data into the table. Note that the final operation will ***move data out of*** /user/data/movielens/cleaned/<datafile> to /user/hive/warehouse/<datafile>. In order to run the script again, you might need to run the specific do_mapreduce.sh script first.

Run the following command. The –e option signifies to Hive that the HiveQL code contained in the string the followed it (SQL from command line).
$ hive –e “describe users”
$ hive –e “select count(*) from ratings”

To print help information type:
$ hive -H

To use the prompt command, type hive and press enter to access Hive shell, where you can do queries interactively.
$ hive
hive>

Now, you can start a query such as
hive> select count(*) from users where gender = 'F';
At the end you will see something like this:
…
OK
1709
Time taken: …

Can you do queries for these questions?
a) How many users are under 35 years old?
b) The top three users who did the most ratings?
c) How many users in each occupation?

Step 4: Build the basic recommender system using Mahout
The last part of this exercise will be to use the data provided in order to make movie recommendations using the analytics tool called Mahout. Mahout provides functions for classification, clustering, recommendations, etc. For this exercise we will only use data from the ratings.dat file in order to provide recommendations. In real life, other features can be used to improve the quality of recommendations, such as age, gender, zip code, preferred movie genres, etc.

a) Prepare data file for Mahout
In order to create the input data file for Mahout, we only need the first 3 columns of data from the ratings table: userid, movieid, rating. We will start with the raw ratings file for this and generate a file with comma separator.

The command for preparing data file for Mahout and take advantage of parallel and distributed of Hadoop for faster processing can be run by typing:

./do_mapreduce_mahout.sh (this invokes the clean_mahout_dat.py python script)

b) Specify users for recommendations
We will also need a file listing the userids that will receive recommendations. Here, the file is called *mahout_users.dat*. We can specify any users we will give the recommendations. For this example, we want the recommendation for user1, user2, and user3. It will look like this:
1
2
3

This file needs to be placed into HDFS for Mahout to see also.

$ hadoop fs -mkdir /user/data/movielens/mahout (create directory before placing the file, if not exists)

$ hadoop fs -put mahout_users.dat /user/data/movielens/mahout

c) Run recommendation using Mahout
Now we are ready to run the recommendation engine. A script is also provided for this.

$ ./do_mahout.sh

If everything is correctly in place, this will run a series of several map-reduce jobs. You can examine the results when finished by typing:

$ hadoop fs -cat /user/data/movielens/mahout/output/part-r-00000

1 [1946:5.0,1012:5.0,1937:5.0,2133:5.0,3655:5.0,617:5.0,832:5.0,1678:5.0,1016:5.0,2190:5.0]
2 [3139:5.0,3862:5.0,1958:5.0,3260:5.0,1879:5.0,1566:5.0,3599:5.0,565:5.0,3614:5.0,1457:5.0]
3 [368:5.0,2372:5.0,3421:5.0,3740:5.0,2048:5.0,2478:5.0,832:5.0,1078:5.0,2857:5.0,1385:5.0]

The numbers in the bracket are the recommended movies (movieid) and its corresponding rating for user1, user2, and user3.

References

Cloudera (n.d.). *Cloudera Documentation*. Retrieved from http://www.cloudera.com/content/support/en/documentation.html.

Grouplens (n.d.). *MovieLens: Data sets*. Retrieved from http://grouplens.org/datasets/movielens/.

Hadoop (n.d.). *Hadoop shell commands.* Retrieved from http://hadoop.apache.org/docs/r0.18.3/hdfs_shell.html.