MSD Homework 1

Jeff Hudson (jdh2182)

Sunday, March 08, 2015

Problem 1

For this problem, the biggest challenge is combining records for pairs of phone numbers in which the caller and callee are reversed. The hack I came up with is to reorder the pairs, always putting the larger number before the smaller. Since the phone numbers are unique and no one can call her/himself, each set of two numbers will always produce the same ordered pair, thus we can be assured that each pair will only be represented once in the final output. The rest of the problem is generating these grouped sums, and figuring out where to incorporate the hack for reordering phone number pairs.

Part A
individuals: 100,000
avg calls: 50
total size of dataset: ~5,000,000 records

Since we have a relatively small dataset (judging by the size of the MovieLens dataset which is 224MB for 10 million records, I am estimating this 5 million record dataset at approximately 100MB), we can hold the entire thing in memory on a single machine and make multiple passes over the data to simplify our calculations. First, we want to reorder all pairs, irregardless of caller or callee. We can simply add a new column (if we're working in `R`, for instance) that concatentates the `caller` and `callee` columns in the following way: if the `caller`'s number is larger than the `callee`'s, it is concatenated first, if not, it is second. For example, if the two numbers in the pair are `2125550123` and `2125559876` then the resulting `pairID` would be `21255598762125550123` regardless of who called whom. Now that we have unique and non-repeated identifiers for each pair, we can use `dplyr` or another implementation of the split-apply-combine approach to group our data by their pairs (`group_by(data,pairID)`) then take the sum of all `call_duration`s for each pair (`summarise(data, sum(call_duration))`).

Part B
individuals: 10,000,000
avg calls: 100
total size of dataset: ~1,000,000,000 records

In this scenario, our dataset is about 200 times bigger (~20GB) and is no longer small enough to hold in RAM. However, we could hold it on a single machine's hard disk and stream through the data once. To hold the solution we will need a large table with two columns, the `pairID`, and the total `call_duration`. This table will initially be empty and will be populated and updated as we stream the dataset. For each row in the dataset, our script will determine the `pairID` as before, by comparing the two numbers, then concatenating them with the larger number first. Then it will find the corresponding row of our output table and add `call_duration` to the total for that row, or if that `pairID` is not yet in our output table, it will create a new entry, and initialize the total with the value of `call_duration` for the current row in the dataset.

Part C
individuals: 300,000,000
avg calls: 200
total size of dataset: ~60,000,000,000 records

At 60 times larger than our previous dataset (~1.2TB), it is no longer feasible to compute these calculations on a single machine, and we need to use a distributed framework on multiple machines such as the `MapReduce` paradigm. Each `Mapper` will receive a portion of the dataset and generate `key-value` pairs as follows: The `key` will be the `pairID` as derived in the earlier parts (`caller` and `callee` phone numbers concatenated with the larger number first). The `value` is simply the `call_duration` for that record. Once all `Mapper`s have finished, the `key-value` pairs will be shuffled and sent to the appropriate `Reducer`s. Each `Reducer` will

receive all `key-value` pairs for some number of `key`s. For each `key` it will add up all the associated values and output the `key` and the sum. Then the outputs of all the `Reducer`s will be collected and we will have a table of our `pairID`s and total `call_duration`s.

Problem 2

See code in `citibike.sh`

```
# count the number of unique stations
tail -n+2 201402-citibike-tripdata.csv | cut -d, -f4 | sort | uniq | wc -l
> 329

# count the number of unique bikes
tail -n+2 201402-citibike-tripdata.csv | cut -d, -f12 | sort | uniq | wc -l
> 5699

# extract all of the trip start times
cut -d, -f2 201402-citibike-tripdata.csv > starttimes.txt
[starttimes are extracted and written to file starttimes.txt; no console output]
[head of starttimes.txt]
> "starttime"
  "2014-02-01 00:00:00"
  "2014-02-01 00:00:03"
  "2014-02-01 00:00:09"
  "2014-02-01 00:00:32"
  "2014-02-01 00:00:41"
  "2014-02-01 00:00:46"
  "2014-02-01 00:01:01"
  "2014-02-01 00:01:11"
  "2014-02-01 00:01:33"

# count the number of trips per day
tail -n+2 201402-citibike-tripdata.csv | cut -d, -f2 | cut -d\  -f1 | tr -d '"' |
  sort | uniq -c
> 12771  2014-02-01
  13816  2014-02-02
   2600  2014-02-03
   8709  2014-02-04
   2746  2014-02-05
   7196  2014-02-06
   8495  2014-02-07
   5986  2014-02-08
   4996  2014-02-09
   6846  2014-02-10
   8343  2014-02-11
   8580  2014-02-12
    876  2014-02-13
   3609  2014-02-14
   2261  2014-02-15
   3003  2014-02-16
   4854  2014-02-17
   5140  2014-02-18
   8506  2014-02-19
  11792  2014-02-20
   8680  2014-02-21
```

```
  13044  2014-02-22
  13324  2014-02-23
  12922  2014-02-24
  12830  2014-02-25
  11188  2014-02-26
  12036  2014-02-27
   9587  2014-02-28

# find the day with the most rides
tail -n+2 201402-citibike-tripdata.csv | cut -d, -f2 | cut -d\  -f1 | tr -d '"' |
  sort | uniq -c | sort | tail -n1
> 13816  2014-02-02

# find the day with the fewest rides
tail -n+2 201402-citibike-tripdata.csv | cut -d, -f2 | cut -d\  -f1 | tr -d '"' |
  sort | uniq -c | sort | head -n1
> 876  2014-02-13

# find the id of the bike with the most rides
tail -n+2 201402-citibike-tripdata.csv | cut -d, -f12 | sort | uniq -c | sort |
  tail -n1
> 130 "20837"

# count the number of riders by gender and birth year
tail -n+2 201402-citibike-tripdata.csv | cut -d, -f14,15 | sort | uniq -c
>     9 "1899","1"
     68 "1900","1"
     11 "1901","1"
      5 "1907","1"
      4 "1910","1"
      1 "1913","1"
      3 "1917","1"
      1 "1921","1"
     32 "1922","1"
      5 "1926","2"
      2 "1927","1"
      1 "1932","1"
      7 "1932","2"
     10 "1933","1"
     21 "1934","1"
     14 "1935","1"
     31 "1936","1"
     24 "1937","1"
     70 "1938","1"
      5 "1938","2"
     24 "1939","1"
     19 "1939","2"
     83 "1940","1"
      1 "1940","2"
    148 "1941","1"
     16 "1941","2"
    173 "1942","1"
      9 "1942","2"
```

```
 108 "1943","1"
  22 "1943","2"
 277 "1944","1"
  34 "1944","2"
 171 "1945","1"
  43 "1945","2"
 424 "1946","1"
  30 "1946","2"
 391 "1947","1"
  60 "1947","2"
 664 "1948","1"
 143 "1948","2"
 624 "1949","1"
 101 "1949","2"
 738 "1950","1"
 152 "1950","2"
   6 "1951","0"
1006 "1951","1"
 146 "1951","2"
1040 "1952","1"
 143 "1952","2"
1474 "1953","1"
 301 "1953","2"
1636 "1954","1"
 306 "1954","2"
1568 "1955","1"
 349 "1955","2"
1777 "1956","1"
 542 "1956","2"
1676 "1957","1"
 562 "1957","2"
2333 "1958","1"
 643 "1958","2"
2281 "1959","1"
 539 "1959","2"
2679 "1960","1"
 776 "1960","2"
2315 "1961","1"
 432 "1961","2"
2808 "1962","1"
 833 "1962","2"
3514 "1963","1"
 715 "1963","2"
3679 "1964","1"
 570 "1964","2"
2957 "1965","1"
 687 "1965","2"
3440 "1966","1"
 565 "1966","2"
4016 "1967","1"
 634 "1967","2"
3931 "1968","1"
 545 "1968","2"
```

```
4557 "1969","1"
 898 "1969","2"
4657 "1970","1"
1079 "1970","2"
4132 "1971","1"
 791 "1971","2"
4066 "1972","1"
 962 "1972","2"
4097 "1973","1"
 877 "1973","2"
4957 "1974","1"
 891 "1974","2"
4185 "1975","1"
 699 "1975","2"
4557 "1976","1"
1022 "1976","2"
4817 "1977","1"
1140 "1977","2"
5645 "1978","1"
1231 "1978","2"
6433 "1979","1"
1338 "1979","2"
6173 "1980","1"
1488 "1980","2"
6620 "1981","1"
1588 "1981","2"
6244 "1982","1"
1724 "1982","2"
6890 "1983","1"
1889 "1983","2"
7348 "1984","1"
1791 "1984","2"
7043 "1985","1"
2262 "1985","2"
6147 "1986","1"
1962 "1986","2"
5776 "1987","1"
1696 "1987","2"
6449 "1988","1"
1599 "1988","2"
5408 "1989","1"
1435 "1989","2"
4541 "1990","1"
1156 "1990","2"
   8 "1991","0"
2377 "1991","1"
 689 "1991","2"
1758 "1992","1"
 410 "1992","2"
1398 "1993","1"
 289 "1993","2"
 927 "1994","1"
 288 "1994","2"
```

```
    664 "1995","1"
    163 "1995","2"
    234 "1996","1"
    100 "1996","2"
    164 "1997","1"
     87 "1997","2"
   6717 \N,"0"

# count the number of trips that start on cross streets that both contain numbers
#   (e.g., "1 Ave & E 15 St", "E 39 St & 2 Ave", ...)
tail -n+2 201402-citibike-tripdata.csv | cut -d, -f5 | awk '/[0-9].*&.*[0-9]/' | wc -l
> 90549

# compute the average trip duraction
tail -n+2 201402-citibike-tripdata.csv | cut -d, -f1 | tr -d '"' |
  awk '{tot += $1} END {print "average = " tot/NR}'
> average = 874.52
```

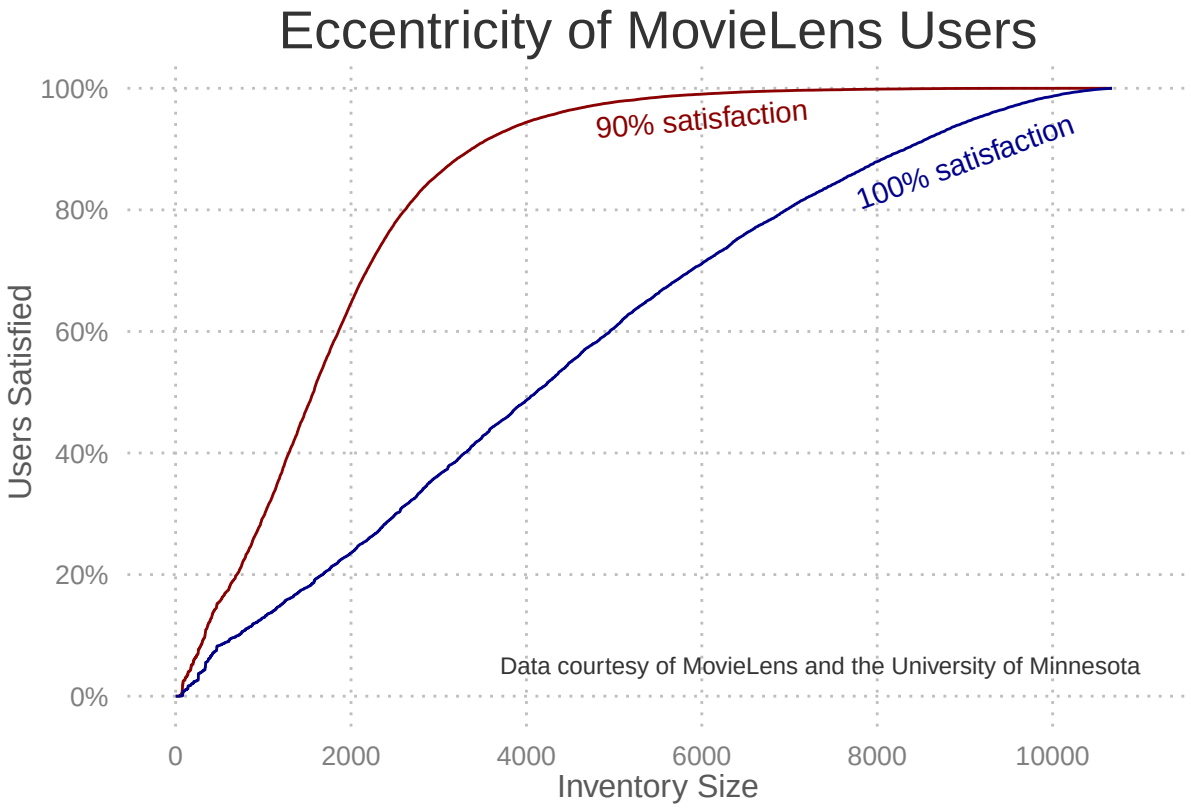
Problem 3

See code in `eccentricity.R`

Solving this problem required several steps. First I read in the dataset, and looked at a few summary statistics: there are `10677` unique movies and `69878` unique users. Clearly it was not feasible to create a sparse matrix with almost 750 million cells. So I needed some other way to store the data compactly but still more usefully than its initial format of one row per user-movie pair. Since the data we eventually want is the number of users who are satisfied at each inventory size, I realized I needed a dataframe with one row per user, and a column containing the list of movies for each user.

Once I had coerced the data to this form, my next strategy was to implement some code by which each user's movie list would be compared to inventories of different sizes to determine how satisfied they were. Ultimately this strategy ended up requiring too much time and space to calculate, however, in the process I created a ranked inventory list which was very useful. I summarized the list of ratings by movieID and summed the number of users who had rated each movie, then sorted the resulting list, this gave me the list of movies in order of popularity. Now I realized that I only needed two things from each user: the inventory size at which they became 90% satisfied, and the inventory size at which they became 100% satisfied.

To obtain these figures, I started by converting their list of rated movies from movieIDs to popularity rank. Thus, each movie was no longer being represented by its ID but by its rank in the popularity list I had created earlier. Once I sorted this list, it became easy to find the movies at which the user became `p`% satisfied: just take `p*n` and round up to the nearest integer to get the index of their movie list which will translate to the smallest inventory size at which they will be `p`% satisfied. Of course, for 100% satisfaction, it is simply the last entry in their movie list. These calculations were much simpler and ran very quickly, which gave me the data needed to create the plot below. To find the number of users satisfied at each inventory size, I simply looped over the dataset and added up the number of users whose [90|100]% satisfiability movie was ranked less than or equal to the inventory size. The rest of the work was just aesthetic tweaking to make the plot visually pleasing and convey the information quickly and easily.


Eccentricity of MovieLens Users
100%
80%
60%
40%
20%
0%
Users Satisfied
90% satisfaction
100% satisfaction
Data courtesy of MovieLens and the University of Minnesota
0
2000
4000
6000
8000
10000
Inventory Size