Systems for Data Science : Milestone 3

Aiday Marlen Kyzy
Sciper : 283505

June 2021

1 Introduction

In this project I have worked on Milestone 3 for the course Systems for Data Science. I have found this project challenging.

2 Optimizing with Breeze

[Question 3.2.1]
The code from the previous milestone has been re implemented using the Breeze library. The MAE I find for $k = 100$ is 0.8351, and for $k = 200$ is 2.9642. The MAE for $k = 100$ is approximately close to 0.7485, however the MAE for $k = 200$ is bigger by a factor of 3. There is likely a small implementation error somewhere in the code.

[Question 3.2.2]
I have used the method `System.nanoTime()` in order to compute the min, max, average and standard deviation of the time required to compute the k-nearest neighbors. These values change at every run of course.

Statistic	Value
Minimum	2805941.299
Maximum	3737574.009
Mean	3034215.386
Standard Deviation	353518.33

The mean time is therefore about 3034215.3862 microseconds or about 3 seconds.

[Question 3.2.3]
I include the same data for the time to compute the predictions :

Statistic	Value
Minimum	5077735.411
Maximum	6197098.600
Mean	5554055.219
Standard Deviation	372553.920

The average is therefore about 5554055.219 microseconds or 6 seconds.

[Question 3.2.4]
In Milestone 2, the mean time for computing the predictions I found was 200401.48 microseconds and the mean time for computing the similarities was 152033.08 microseconds. In fact my implementation with Breeze requires more time than my previous implementation. This is perhaps because I was persisting RDDs and therefore not performing the same calculation multiple times. I also tried to optimize my code in Milestone 2, hence why I had relatively low execution times. In this case I'll calculate the reverse speedup $\frac{t_{new}}{t_{old}} = \frac{3034215.386}{152033.08} \approx 20$ for the similarities, and $\frac{t_{new}}{t_{old}} = \frac{5554055.219}{200401.48} \approx 27$.

3 Scaling with Spark

[Question 4.1.1]
I now train my implementation with the `ml-1m/ra.train` dataset and I test with the `ml-1m/ra.test` set. The MAE I obtain is 0.7233. It is not exactly equal to the MAE given 0.7485, but it is quite close. It is also closer to the actual MAE than in the previous optimization section. This means there must be a small difference in the implementations.

[Question 4.1.2]
I run my implementation 5 times on the cluster, and I get the following data when using one executor and running locally with `--master "local[1]"` :

Run number	KNN time in microseconds	Prediction time in microseconds
1	925187282.273	1006954795.890
2	918528555.541	998466918.041
3	939009630	1018345595
4	976174591	1042685114
5	941993668	1024069180

The mean of the running times to compute the KNN is 940178745.363 microseconds or 16 minutes. The mean of the running times to compute the predictions is 1018104320.59 microseconds or 17 minutes. Now the code from section 3 was not made to accommodate the bigger set `ml-1m/ra.train`. In section 3 for example, I have implemented matrix/matrix products which would give an OutOfError error if I were to run the bigger dataset on this code. However we still notice that on the dataset `ml-100k/u1.base` in section 3, the average execution time to compute the similarities is about 3 seconds and to compute the predictions is about 6 seconds. Now since our current data set is about 10 times bigger than the previous data set, we expect to require about 10 times more computation time on the bigger data set. However this is not the case as $3 \cdot 10 << 16 \cdot 60$ and $6 \cdot 10 << 17 \cdot 60$. Therefore the new code adds a lot of overhead. This implementation could be made more quick if instead we could use matrix/matrix multiplications as was done in the previous section, instead of splitting the code into vector/matrix and vector/vector multiplications. This is of course not always possible as was the case here, if we have limited storage space.

[Question 4.1.3]
The running times for 1 executor for 3 trials are as follows :

Run number	KNN time in microseconds	Prediction time in microseconds
1	925187282.273	1006954795.890
2	918528555.541	998466918.041
3	939009630	1018345595

The average for the similarities is : 927575155.938. The average for the predictions is 1007922436.31. For 2 executors :

Run number	KNN time in microseconds	Prediction time in microseconds
1	463812468	590247892
2	513374083	625620082
3	443492554	563342646

The average for the similarities is : 473559701.667. The average for the predictions is : 593070206.667. For 4 executors :

Run number	KNN time in microseconds	Prediction time in microseconds
1	244906032	303354199
2	225751390	285470138
3	243467961	308496847

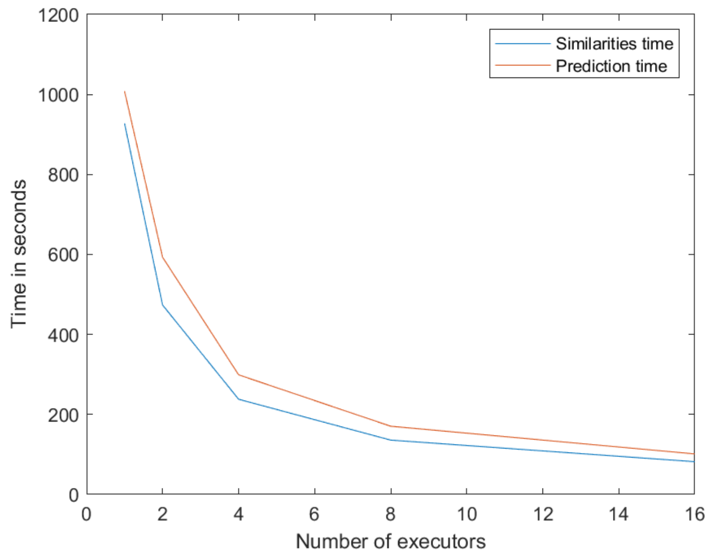
The average for the similarities is 238041794.333. The average for the predictions is 299107061.333. For 8 executors :

Run number	KNN time in microseconds	Prediction time in microseconds
1	141608933	175880018
2	133972603	168774498
3	132187691	167174277

The average for the similarities is 135923075, and the average for the prediction times is 170609598. For 16 executors I find the following data :

Run number	KNN time in microseconds	Prediction time in microseconds
1	84329729	104437159
2	83562010	103215248
3	77907771	96539167

The average for the similarities is 81933170, the average for the predictions is 101397191.333. From the data in the tables above we are able to draw the following graph for the average time for the similarities and for the predictions in seconds which depends on the number of executors:



From the graph above we see that adding executors in the beginning decreases the execution time approximately by a factor of two. Indeed the ratio of the execution times to compute the similarities when 1 executor is used versus 2 is $\frac{927575155.938}{473559701.667} = 1.96$. As you add more executors, the execution time decreases less and less. Indeed we see that going from 8 to 16 executors, the execution time to compute the similarities decreases only by a factor of $\frac{135923075}{81933170} = 1.65$. Similarly the ratio of the execution times to compute the predictions is $\frac{170609598}{101397191.333} = 1.68$. This means that adding more and more executors starts having a more marginal effect. The execution time grows linearly only when a small number of executors is used.

4 Economics

[Question 5.1.1]
The ICC.M7 costs 35'000 CHF according to the assumption. We calculate the cost of renting. Presumably if you rent then that means you only pay the operating costs every day so you pay 20.40 CHF/day. To make it less expensive to rent you need to wait 35'000/20.40 $\approx$ 1716 day. This is equivalent to 4.7 years.

[Question 5.1.2]
The ICC.M7 costs 20.40 CHF/day. Now if we need 1536 GB of RAM, then this translates to $1536 \cdot 0.012 = 18.432$ CHF/day for the container. The ICC.M7 has 28 cores, which translates to $28 \cdot 0.092 = 2.576$ CHF/day. The ratio is therefore : $\frac{20.40}{21.008} \approx 0.971$. The containers are a little more expensive than the ICC.M7 but not significantly more expensive.

[Question 5.1.3]
Suppose we have 4 Raspberry Pis. Then the rent per day at max power is $4 \cdot 0.0504 = 0.2016$. Now $32 = 4\dot{8}$ GB for the containers costs $32 \cdot 0.012 = 0.384$ per day. We have a total of 16 Cortex-A72 cores. For the containers, this is $16 \cdot 0.092 = 1.472$ CHF/day. The ratio is then : $\frac{0.2016}{1.856}$. When Raspberry Pi is operating at lowest power, the ratio is $\frac{0.0432}{1.856}$. In both cases the ratio is low, hence the Raspberry Pis are cheaper.

[Question 5.1.4]
Suppose we buy and run 4 Raspberry Pi devices. After t days we will have paid : $4 \cdot 94.83 + 0.0432 \cdot t$ when running at minimum power and $4 \cdot 94.83 + 0.2016 \cdot t$ when running at maximum power. We previously calculated that per day the container costs 1.856 CHF/day if we have the same CPU and RAM requirements. This means that after t day we would spend $1.856 \cdot t$ CHF. In the case of maximum power, the cost becomes higher when the following equation is satisfied : $4 \cdot 94.83 + 0.2016 \cdot t = 1.856 \cdot t$, that is after $t > 4 \cdot 94.83/1.6544 = 229.3$ days. That is after 230 days. In the case of minimum power, we need $t > 4 \cdot 94.83/1.8128 = 209.2$. That is after 210 days.

[Question 5.1.5]
The ICC.M7 costs 35'000 CHF. With this money, one could buy $35'000/94.38 \approx 369$ Raspberry Pis. The throughput does not increase with more Raspberry Pis, it remains at 1.5 GHz. The ratio is therefore $1.5/2.6 = 0.58$. Concerning the RAM however, it increases to $8 \cdot 369 = 2952GB = 2.952TB$. We get more RAM by a factor of $2.952/1.536 = 1.92$.

[Question 5.1.6]
According to the assumptions we need to keep in memory the ratings and the similarity values. Each user is assigned 200 similarities and 100 ratings, so we need to store 200 floating point numbers ($32 \cdot 200 = 6400$ bits) and 100 unsigned integers (800 bits). In total we store 7200 bits per user. Suppose we have 1 GB and we can only allocate 0.5 GB for these similarities and ratings, then we can store $0.5 \cdot 10^9 \cdot 8/7200 = 555'555$ users. Suppose instead we have a Raspberry Pi, then we have 8GB of memory, and hence we can store $8 \cdot 555'555 = 4'444'440$ users. Supposing we have an ICC.M7 then we can store $0.5 \cdot 1.536 \cdot 10^{12} \cdot 8/7200 = 853'333'333$ users.

[Question 5.1.7]
I would prefer to buy Raspberry Pis as they are cheaper than containers and the ICC.M7 and you can achieve great storage capacity, by buying lots of them. If one breaks, it is possible to easily change them and maintain the infrastructure.

5 Conclusion

This was an interesting project, though quite difficult in that I had to deal with the OutOfMemory error quite often and had to optimize the code. I have not been able to obtain the correct values for the MAE, and this is indicative that there is an implementation error somewhere. I have worked a lot on the project, and found it quite challenging.