ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

PROJECT MILESTONE 3: OPTIMIZING, SCALING, AND ECONOMICS

CS-449 SYSTEMS FOR DATA SCIENCE

Nour Ghribi, Sciper: 250232

23rd September 2021

As before, for making the code more modular there is package object called utils (can be found in src/main/scala/{optimizing,scaling}/utils.scala) that contains all the functions used and relevant methods. (Some functions and methods of the previous milestones have been kept and adapted with new additional ones for this milestone.)
All the functions defined in utils have their respective description that can be generated as scala doc by the doc command of sbt. "sbt:m1_250232> doc"

Contents

CHAPTER 1

MILESTONE 1

1.1 ANSWERS:

Q.3.1.1

On average the ratings do not coincide with the middle of the $(1, 2, 3, 4, 5)$ scale, the average rating is approximately 3.53 so on average ratings are higher by 0.53 than the middle of the scale.

Q.3.1.2

We can see that there is quite a disparity in users average ratings. Some are quite low compared to the global average rating and some tend to have a larger average than the global average thus rate more positively.
The average of the average ratings per user is ≈ 3.59, the minimum is ≈ 1.5 and the maximum is ≈ 4.87 .
Considering ratings that deviate with less than 0.5 from the global average close: The ratio of users that are close to the global average is: ≈ 0.75 that is, we have a $3 : 4$ ratio approximately of users average ratings that are close to the global rating.

Q.3.1.3

Looking at the data grouped by items, we have an average of the average ratings per item ≈ 3.08 which is quite close to the middle of the scale. However, not all items are close to the global average. We have a ratio of ≈ 0.49 items whose average ratings are close to the global average -> almost half of the items have an average close to the global average.

Q.3.1.4

Comparing averages predictions to the baseline:

Model	MAE
Global Average	0.9680
Users Averages	0.8501
Items Averages	0.8275
Baseline	0.7669

- We can see that the baseline model outperforms all of the other prediction models based on the averages.

- The highest MAE can be seen using the global average method and it is logical as the global average can not be generalized on all the users ratings patterns or the success of some items.
- Using the user ratings averages we improve our prediction but still suffer an important loss. Some users tend to have a disparity in their ratings therefore this method that introduces a bias into the predictions doe not work so well. In addition, some items are quite successful that they will have a good rating regardless of the user average. These items will maybe have a positive deviation from the user's average rating.
- Proceeding with the items rating averages as predictors, we discover an improvement from the users rating averages method. We can interpret this result as: an item's average is more logical to have as predictor than the user's average, as this will have the item's bias instead of the user's and thus will give more information about the item in question.
- The baseline model is the best predictor so far. Based on the items average deviations that incorporates the users biases, this method takes into account the success of each item compared to the different rating patterns of users. It is only logical that it would perform the best out of the 4 methods.

Q.3.1.5

Technical specifications:
- MacbookPro (13-inch, 2016, Four Thunderbolt 3 Ports)
- 2,9 GHz Intel Core i5 double cœur
- 8 Go 2133 MHz LPDDR3
- MacOS Big Sur 11.2.3
- sbt 1.4.7 (JetBrains s.r.o Java 11.0.6)

Execution times in microseconds				
Model	Minimum	Maximum	Average	Standard Deviation
Global Average	201905.09	884357.39	482547.5	253169.03
Users Averages	521201.12	3336583.46	1320295.57	971568.55
Items Averages	494299.17	862203.12	703579.93	100270.5
Baseline	2635555.89	4101214.77	3353147.37	437269.39

- Looking at the average execution times, we can see that the most expensive method is the baseline model. It follows from the fact that it uses the items average deviations and user averages.
- The fastest model is the global average method followed by the users averages and then the items averages model.
- We have a ratio of ≈ 6.95, that is the global average method is approximately 7 times faster than the baseline model. This result shows the consensus one has to make between performance times and correctness.

Q.4.1.1

Adding personal ratings on movies already seen in personal.csv, and fitting the model on the expanded data, some of the recommended movies might interest me to watch. However, these results show that these recommendations might come from the fact that these movies are not rated that much but have a full score of 5.0 on only one rating.

The recommendations are :

1. 814, "Great Day in Harlem", 5.0
2. 1122, "They Made Me a Criminal (1939)", 5.0
3. 1189, "Prefontaine (1997)", 5.0
4. 1201, "Marlene Dietrich: Shadow and Light (1996)", 5.0
5. 1293, "Star Kid (1997)", 5.0

Q.4.1.2

To favour more popular movies, it seemed logical to think about a mapping for "being famous" and a weight on the predictions. The solution is to consider a sigmoid on the count of number of ratings of movies with a "famous criteria" call it f_c. The new model would be:

$$p_{u,i} = w_i * (\bar{r}_{u,\bullet} + \bar{\hat{r}}_{\bullet,i} * scale((\bar{r}_{u,\bullet} + \bar{\hat{r}}_{\bullet,i}), \bar{r}_{u,\bullet}) - 1) + 1$$

with

$$w_i = sigmoid(count_i - f_c),\ sigmoid(x) = \frac{1}{1 + \exp(-x)}$$

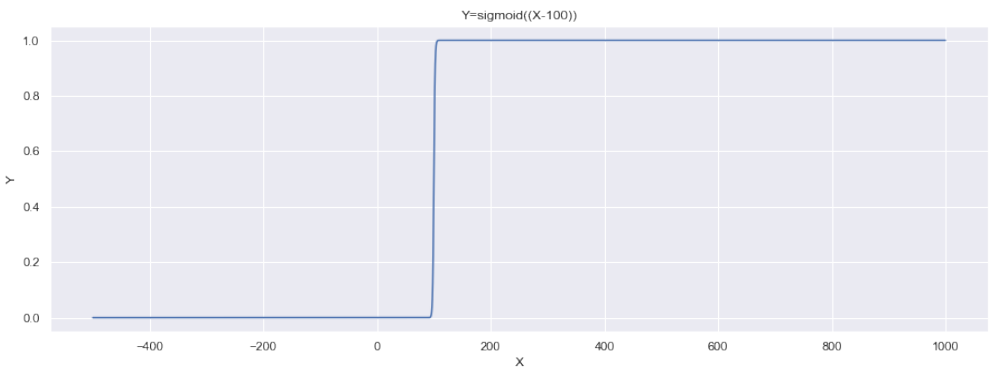
and $count$ the number of ratings of item i.

The idea is to shift the sigmoid on the "famous criteria" f_c, that is the count of which we will consider a movie famous and associate this value to how important the prediction is. It follows that multiplying the baseline by this value gives a prediction that takes into account the popularity. This model will favor more famous movies to have almost the same prediction and penalizes the less popular movies. Some scaling by constants was necessary to keep the predictions in the range $[1, 5]$.

1.2 APPENDIX

1.2.1 SIGMOID FUNCTION:

FIGURE 1.1
Y=Sigmoid(X-100)



CHAPTER 2

MILESTONE 2

2.1 ANSWERS:

Q.2.2.1

Adjusted-Cosine Metric: Using the adjusted-cosine similarity based model we get an MAE of ≈ 0.748 which has a difference of ≈ -0.019 with the baseline. So this model is better than the baseline model of last milestone.

Q.2.2.2

Jaccard Metric: We will take two sets A_u and B_v corresponding to the set of items rated by user u and user v respectively.

Our Jaccard metric will be :

$$J(u,v) = \frac{|A_u \cap B_v|}{|A_u \cup B_v|} = \frac{|A_u \cap B_v|}{|A_u|+|B_v|-|A_u \cap B_v|}$$

Jaccard MAE ≈ 0.762

Jaccard MAE - Cosine MAE = ≈ 0.014

We see that the Jaccard based prediction is worst than the adjusted cosine based one given their MAE.

Q.2.2.3

- We consider the similarity of users with themselves
- We don't take into account the symmetric aspect of similarities to reduce the size of the similarities. ($s_{u,v} = s_{v,u}$)

Given user data U, if every user in the dataset rated at least one other item in common with all the other users, we'll have to compute $|U|*|U|$ similarities. In our case for the ml-100k data (u1.base) : $|S_{\bullet,\bullet}| = |U|*|U| = 889249$

If we consider $u \in U$ and $v \in V$, with U the users of the **test set** and V the users of the **train set**: $|S_{\bullet,\bullet}| = |U|*|V| = 432837$. The latter is how this implementation computed the similarities because it seemed logical at first but we will include both situations for each question and the code uses only the train data for computing **all** the similarities.

Q.2.2.4

Based on the adjusted-cosine for similarities and given that the number of multiplications needed to compute a similarity between u and v is the same as $|I(u) \cap I(v)|$:

If we take $u \in U$ and $v \in V$, with U and V the users of the **train set** we get:

Cosine Similarity Statistics	
Statistic	value
Min	0
Max	685
Average	18.175
Standard Deviation	15.257

So the minimum number of multiplications is 0 (when a user has no other similar user) and maximum 685.

If we take $u \in U$ and $v \in V$, with U the users of the **test set** and V the users of the **train set** we get:

Cosine Similarity Statistics	
Statistic	value
Min	0
Max	582
Average	9.888
Standard Deviation	15.257

So the minimum number of multiplications is 0 (when a user has no other similar user) and maximum 582.

Q.2.2.5

The memory needed for storing all $s_{u,v}$ is $8 * |S_{\bullet,\bullet}| = 8 * |U| * |U|$ bytes

Considering all the users present in our train data U with all the other users present in our train data this gives $8 * |S_{\bullet,\bullet}| = 8 * |U| * |U|$ bytes :

- Total bytes to store all similarities: 7113992 Bytes
- Total bytes to store non-zero similarities: 6592280 Bytes

Considering only the users present in our test data with all the users present in our train data this gives $8 * |S_{\bullet,\bullet}| = 8 * |U| * |V|$ bytes ($U \rightarrow$ u1.base , $V \rightarrow$ u1.test) :

- Total bytes to store all similarities: 3462696 Bytes
- Total bytes to store non-zero similarities: 3128496 Bytes

Q.2.2.6

Measuring the time required for computing predictions (with 1 Spark executor), including computing the similarities $s_{u,v}$.

Based on U and V the users of the **train set**:

Execution times in microseconds	
Statistic	value
Min	50161739.491
Max	62476757.833
Average	54086555.463
Standard Deviation	4347867.0933

Based on U the users of the **test set** and V the users of the **train set**:

Execution times in microseconds	
Statistic	value
Min	28088031.566
Max	30729199.976
Average	29462053.126
Standard Deviation	902355.131

Milestone 1, Q.3.1.5

Execution times in microseconds				
Model	Minimum	Maximum	Average	Standard Deviation
Global Average	201905.09	884357.39	482547.5	253169.03
Users Averages	521201.12	3336583.46	1320295.57	971568.55
Items Averages	494299.17	862203.12	703579.93	100270.5
Baseline	2635555.89	4101214.77	3353147.37	437269.39

This method is computationally expensive compared to the baseline of the last milestone. The average is way bigger than the baseline which was the most expensive one, this is due to the fact that we are computing the similarities between all the users present in the test set with all the users in the train set. Which is more expensive if we computed all the ones present in the train set.

Q.2.2.7

Measuring only the time for computing similarities (with 1 Spark executor).

Based on U and V the users of the **train set**:

Execution times in microseconds	
Statistic	value
Min	23594698.211
Max	6523980.075
Average	25701488.659
Standard Deviation	1077605.993

The average time in micro second per $S_{u,v} \approx 59.379 \mu s$ and we have a ratio between time to compute similarity over time to predict ≈ 0.48.

Based on U the users of the **test set** and V the users of the **train set**:

Execution times in microseconds	
Statistic	value
Min	13551911.518
Max	15021283.858
Average	14287601.055
Standard Deviation	478555.242

The average time in micro second per $S_{u,v} \approx 33\mu s$ and we have a ratio between time to compute similarity over time to predict ≈ 0.5.

- We can see by the ratio of the time to compute the similarities over time to predict that computing the similarities takes almost half of the running time for running the predictions.
- Considering only the users present in the test to compute their similarities speeds up the runtime.

Q.3.1.1

KNN - Keeping the k closest users based on similarity

As we vary k we get better predictions based on the MAE until getting to an upper bound which when we get past the predictions get worse.

This upper bound is reached at an MAE of 0.747 for $k \in \{300, 400, 800\}$. The smallest k for which we predict better than the baseline is 100.

We get an MAE of 0.756 which is ≈ 0.011 lower of the baseline.

MAE per value of k	
k	MAE
10	0.842
30	0.793
50	0.775
100	0.756
200	0.749
300	0.747
400	0.747
800	0.747
943	0.748

Q.3.1.2

Given that $u \in U$ and $v \in V$, with U and V the users of the **train set**: $|S_{\bullet,\bullet}| = |U| * |V| = 889249$.

Now, if we take only the k closest neighbours (k biggest similarities) for each user, we'll have to store $|U| * k$ similarities. So the minimum number of bytes required to store the k closest neighbours for each user u is : $8 * |U| * k$ which gives for each values of k:

Memory for values of k	
k	Bytes needed
10	75440
30	226320
50	377200
100	754400
200	1508800
300	2263200
400	3017600
800	6035200
943	7113992

As before, if we assume that we work with $u \in U$ and $v \in V$, with U the users of the **test set** and V the users of the **train set**, hence: $|S_{\bullet,\bullet}| = |U| * |V| = 432837$, we have $|U| = 459$ and $8 * |U| * k$ is needed to store the similarities for each user :

Memory for values of k	
k	Bytes
10	36720
30	110160
50	183600
100	367200
200	734400
300	1101600
400	1468800
800	2937600
943	3462696

Q.3.1.3

The RAM is 8GB = 8 000 000 000 Bytes. Given that for each similarity we need 2 integers for indices (64-bit each one) and we store it as double (64-bit), we need $3 * 8 = 24$ bytes for each entry.

Given that we chose $k = 100$, we'll have $k * |U|$ Similarities, taking into account that we take only u of the test set (u1.test) we'll be able to fit a maximum number of $\lfloor \frac{8GB}{k*3*8} \rfloor$ users. -> 3333333 users

Q.3.1.4

Varying k has no impact on the number of the similarities to compute as we need to compute all the similarities for each user u and then search for it k nearest neighbours v's based on this metric.

Q.3.1.5

Recommendations Keeping the personal ratings on movies already seen in personal.csv the same as of the last milestone, and fitting the model on the expanded data, some of the recommended movies interest me to watch and are better then the ones of the last milestones. They are more inline with the movies I rated on the personal file and since my ratings aren't that diverse (Most are > 3.5) this may have lead to not so similar users influencing the prediction, adjusting the personal ratings givesthe new movies cited afterwards. The movies differ for $k = 30$ and $k = 300$ and this is only logical as with the latter we have more similarities that influence the weighted sum. The recommendations are :

For $k = 30$:

1. 135, "2001: A Space Odyssey (1968)", 5.0
2. 152, "Sleeper (1973)", 5.0
3. 162, "On Golden Pond (1981)", 5.0
4. 170, "Cinema Paradiso (1988)", 5.0
5. 211, "M*A*S*H (1970)", 5.0

For $k = 300$:

1. 113, "Horseman on the Roof", 5.0
2. 119, "Maya Lin: A Strong Clear Vision (1994)", 5.0
3. 814, "Great Day in Harlem", 5.0
4. 935, "Paradise Road (1997)", 5.0

5. 958, "To Live (Huozhe) (1994)", 5.0

Adjusted Personal Ratings Recommendations Result:

For $k = 30$:

1. 48, "Hoop Dreams (1994)", 5.0
2. 134, "Citizen Kane (1941)", 5.0
3. 136, "Mr. Smith Goes to Washington (1939)", 5.0
4. 152, "Sleeper (1973)", 5.0
5. 169, "Wrong Trousers", 5.0

For $k = 300$:

1. 113, "Horseman on the Roof", 5.0
2. 119, "Maya Lin: A Strong Clear Vision (1994)", 5.0
3. 814, "Great Day in Harlem", 5.0
4. 935, "Paradise Road (1997)", 5.0
5. 954, "Unzipped (1995)", 5.0

CHAPTER 3

MILESTONE 3

3.1 ANSWERS:

3.1.1 OPTIMIZING:

Q.3.2.1

Reimplementationn of the KNN with Breeze Library. (Code)

Q.3.2.2

Time for computing k-nearest neighbours on the data "ml-100k/u1.base"

Execution times in microseconds	
Statistic	value
Min	445316.124
Max	845878.782
Average	527362.91
Standard Deviation	116675.08

Q.3.2.3

Time for computing predictions on data "ml-100k/u1.test" after computing k-nearest neighbours on the data "ml-100k/u1.base":

Execution times in microseconds	
Statistic	value
Min	2397545.428
Max	5233575.004
Average	3772880.124
Standard Deviation	1118200.299

Q.3.2.4

Compared to the previous implementation usinng RDDs, this implementation using the Breez library and CSCMatrix sparse matrices and linear algebra functions got us a speed up of:

$$\frac{t_{old}}{t_{new}} = \frac{25701488.659}{527362.91} = 48.73$$

that is our implementation for knn is $\approx 49x$ faster !

3.1.2 SCALING:

Q.4.1.1

The MAE for the ml-1m dataset with k = 200 is 0.7346

Q.4.1.2

TABLE 3.1
Computation Time with Spark for knn and prediction in seconds

	1	2	3	4	5	Mean
knn	18.634	18.453	19.183	18.932	19.347	18.910
Prediction	100.425	100.851	116.683	114.928	120.181	110.614

TABLE 3.2
Computation Time without Spark for knn and prediction in seconds

	1	2	3	4	5	Mean
Knn	23.296	24.738	25.251	28.395	25.086	25.322
Prediction	140.807	138.123	149.898	145.659	134.448	141.787

Q.4.1.3

The results are summarized in Table 3.3 and Table 3.4. The speedup increasing with the number of executors. The gain is linear in the first steps but start to slowdown with more than 8 executors. This decrease can be explained by the fact that every partition is loading the required data, which at some point, worsen overall performance

TABLE 3.3
Prediction time in seconds for different workers

#workers	1	2	4	8	16
Min	91.139	46.152	26.411	15.209	8.126
Max	98.353	48.649	27.172	15.933	8.653
Mean	94.784	47.548	26.715	15.475	8.427

TABLE 3.4
Knn time in seconds for different workers

#workers	1	2	4	8	16
Min	14.072	7.128	4.291	3.111	1.985
Max	19.830	11.049	7.592	6.173	5.368
Mean	16.788	8.869	5.589	4.176	3.217

3.1.3 ECNOMICS:

Q.5.1.1

The ICC.M7 costs $35000CHF$ to buy and $20.4CHF$ per day for renting. If we buy it that is the equivalent of renting it for $\frac{35000}{20.4} \approx 1716$ days which is: $\frac{1716}{365} = 4.7$ so roughly 5 years.

Q.5.1.2

Daily cost of an IC container with the same specs as the ICC.M7($1vCPU = 1Core$):

Cost of IC container:

RAM: $24 * 64 * 0.012 = 18.432$ CHF/day

CPU: $2 * 14 * 0.088 = 2.464$ CHF/day

Total: 18.432 + 2.464 = 20.896 CHF/day

The ratio of the rent of the ICC.M7 with its equivalent container is $(\frac{costICC.M7}{costCONTAINER})$:

$$\frac{20.4(CHF/day)}{20.896(CHF/day)} = 0.976$$

ANSWER: So Renting the container turns out to be more expensive than renting the ICC.M7.

Q.5.1.3

Container vs **4 RPis**: 1vCPU = 1 Intel core = 4 RPis in throughput annd same amount of RAM:

Cost of equivalent container of 4 Raspberry Pis:

RAM: $4 * 8 * 0.012 = 0.384$ CHF/day

CPU: 0.088 CHF/day

Total: 0.384 + 0.088 = 0.472 CHF/day

Cost of 4 Raspberry Pis at max power:

$4 * 0.054 = 0.216$ CHF/day

Cost of 4 Raspberry Pis at min power:

$4 * 0.0108 = 0.0432$ CHF/day

Ratio of operating costs of 4 Raspberry with their equivalent container is $(\frac{cost(4*RPi4)}{costCONTAINER})$:

At max power:

$$\frac{0.216(CHF/day)}{0.472(CHF/day)} = 0.457$$

At min power:

$$\frac{0.0432(CHF/day)}{0.472(CHF/day)} = 0.091$$

ANSWER: Renting the container turns out to be more expensive then operating the 4 Raspberry Pis.

Q.5.1.4

Cost for buying + running 4 Raspberry Pis 4:

Buying: $4 * 94.83 = 379.32$ CHF

Min energy running cost: $4 * 0.0108 = 0.0432$ CHF/day

Max energy running cost: $4 * 0.054 = 0.216$ CHF/day

Thus to answer the question let d be the number of days that the container is rented or the 4 RPis are operated: **Cost of running container:** $0.472 * d$

Cost of buying and running the 4 RPis:

$(379.32 + 0.0432 * d, 379.32 + 0.216 * d)$ with (min_energy, max_energy)

Thus:

Min days for renting container becomes more expensive then buying and running 4 RPis:

Minimum energy:

$$0.472 * d > 379.32 + 0.0432 * d \implies d > \frac{379.32}{0.472 - 0.0432}$$

Maximum energy:

$$0.472 * d > 379.32 + 0.216 * d \implies d > \frac{379.32}{0.472 - 0.216}$$

ANSWER: Minimum days for renting the container so that the cost is higher than buying and running 4 Raspberry Pis: $885 days$ for minimum energy and $1482 days$ for maximum energy.

Q.5.1.5

We can get 369 Raspberry Pis for the buying price of the ICC.M7.

Thus 369 Raspberry Pis we'll give us a throughput of : $\frac{369}{4} = 92.25$ vCPUs and $369 * 8GB = 2952GB$ of RAM.

Ratios (RPis 4 vs ICC.M7):

CPU Thoughput ratio: $\frac{92.25}{28} = 3.295$

Ram ratio: $\frac{2952GB}{(24*64)GB} = 1.922$

Q.5.1.6

We have onnn average 100 ratings per user with 8 bit unigned ints = 1 bytes 100 bytes in ratings per User. Best k for k-nn is k=200, similarities use 32-bit floating points = 4 bytes 800 bytes in similarities per User. Thus each user has 800+100 = 900 bytes of data.

Given that each hardware offer has to leave 50% of memory off:

- UsersPerGb: 555555.0
- UsersPerRPi: 4444440.0
- UsersPerICCM7: $8.533 * 10^8$

Q.5.1.7

Given all the results, the best option would be: to buy 4 Raspberry Pis and run the model on them, because:

- In the long run this is more profitable than running the model on the equivalent container.

- It is the cheapest option. If we take into acount the buying price of the 4 Raspberry Pis 4 Compute models which is $379.32CHF$, we can break even with just $1.039CHF/day$ profit in one year or with $4.168CHF/day$ profit in 3 months.
- We can fit $4 * 10^6$ users with their ratings and similarities so this can be enough for offering a good experience, and in the case of potential more scaling we can always add more Raspberry Pis.
- **It is to note** that we will have to maintain the Raspberry Pis modules and invest labor time, we can always opt for the containers and aim for more profit to break even.

To conclude, we will have to make a consensus, between investing much more money and having the ease of use of containers or servers and much more RAM or investing more time and effort and make the most of cheap powerful tools such as the Raspberry Pis 4 compute models.