CS-449 Project Milestone 2
Optimizing, Scaling, and Economics

Name	Fedor Sergeev
Sciper	323636
Email	fedor.sergeev@epfl.ch

Name	Ivan Korovkin
Sciper	342338
Email	ivan.korovkin@epfl.ch

GitHub: github.com/TheodorSergeev/knn-movie-recommender-2

May 20, 2022

For convenience our comments are given in this color.

1 Optimizing with Breeze, a Linear Algebra Library

BR.1 Reimplement the kNN predictor of Milestone 1 using the Breeze library and without using Spark. Using $k = 10$ and `data/ml-100k/u2.base` for training, output the similarities between: (1) user 1 and itself; (2) user 1 and user 864; (3) user 1 and user 886. Still using $k = 10$, output the prediction for user 1 and item 1 ($p_{1,1}$), the prediction for user 327 and item 2 ($p_{327,2}$), and make sure that you obtain an MAE of 0.8287 ± 0.0001 on `data/ml-100k/u2.test`.

The following Table 1 contains the information about similarities between user 1 and itself; user 1 and 864; user 1 and 886; prediction for user 1 and item 1; prediction for user 327 and item 2; MAE.

$s_{1,1}$	0
$s_{1,864}$	0.2423
$s_{1,886}$	0
$p_{1,1}$	4.3191
$p_{327,2}$	2.6994
MAE	0.8287

Table 1: Optimizing results for $k = 10$.

BR.2 Try making your implementation as fast as possible, both for computing all k-nearest neighbours and for computing the predictions and MAE on a test set. Your implementation should be based around `CSCMatrix`, but may involve conversions for individual operations. We will test your implementation on a secret test set. The teams with both a correct answer and the shortest time will receive more points.

Using $k = 300$, compare the time for predicting all values and computing the MAE of `ml-100k/u2.test` to the one you obtained in Milestone 1. What is the speedup of your new implementation (as a ratio of $\frac{average\ time_{old}}{average\ time_{new}}$)? Use the same machine to measure the time for both versions and provide the answer in your report.

Also ensure your implementation works with `data/ml-1m/rb.train` and `data/ml-1m/rb.test` since you will reuse it in the next questions.

Current implementation increases the time for computing mean absolute error by a factor of ${}^{526.8}/_{3.0} = 175.6$. The time measurements, made on IC cluster, are shown in the following Table 2.

Implementation	Time, s
Milestone 1	527
Milestone 2	3

Table 2: KNN time of different Milestones (laptop) (x175 speedup).

2 Parallel k-NN Computations with Replicated Ratings

EK.1 Test your parallel implementation of k-NN for correctness with two workers. Using $k = 10$ and `data/ml-100k/u2.base` for training, output the similarities between: (1) user 1 and itself; (2) user 1 and user 864; (3) user 1 and user 886. Still using $k = 10$, output the prediction for user 1 and item 1 ($p_{1,1}$), the prediction for user 327 and item 2 ($p_{327,2}$), and make sure that you obtain an MAE of 0.8287 ± 0.0001 on `data/ml-100k/u2.test`

The following Table 3 contains the information about similarities between user 1 and itself; user 1 and 864; user 1 and 886; prediction for user 1 and item 1; prediction for user 327 and item 2; MAE.

$s_{1,1}$	0
$s_{1,864}$	0.242
$s_{1,886}$	0
$p_{1,1}$	4.319
$p_{327,2}$	2.699
MAE	0.828

Table 3: Parallel KNN results.

EK.2 Measure and report the combined *k-NN* and *prediction* time when using 1, 2, 4 workers, $k = 300$, and `ml-1m/rb.train` for training and `ml-1m/rb.test` for test, on the cluster (or a machine with at least 4 physical cores). Perform 3 measurements for each experiment and report the average and standard-deviation total time, including training, making predictions, and computing the MAE. Do you observe a speedup? Does this speedup grow linearly with the number of executors, i.e. is the running time X times faster when using X executors compared to using a single executor? Answer both questions in your report.

The following Table 4 represents acquired results on 1m dataset with varying number of workers and k=300, executed on IC cluster.

Executors	time, s	StDev, s
1	58.4	0.9
2	51.3	0.2
4	45.1	0.7

Table 4: Time measurements of parallel KNN.

The speed up is observed when increasing the number of executors. In the first version of the Breeze-powered KNN that we implemented, we didn't convert slices of CSCMatrix using toDenseVector, and we had an even more significant increase in performance when increasing the number of executors: 1 executor - 400 sec, 2 executors - 300 sec, 4 executors - 220 sec. With the conversion to a DenseVector, the speedup is not so dramatic, but the solution runs significantly faster.

Currently, the speedup does grow linearly with a factor of approximately 1.13. In the previous implementation without toDenseVector, when the overall running time was higher, the factor was approximately 1.3.

3 Distributed Approximate k-NN

AK.1 Implement the approximate k-NN using your previous breeze implementation and Spark's RDDs. Using the partitioner of the template with 10 partitions and 2 replications, $k = 10$, and `data/ml-100k/u2.base` for training, output the similarities of the approximate k-NN between user 1 and the following users: $1, 864, 344, 16, 334, 2$.

The following Table 5 contains the information about similarities between user 1 and users $1, 864, 344, 16, 334, 2$.

$s_{1,1}$	0
$s_{1,864}$	0
$s_{1,344}$	0.236
$s_{1,16}$	0.185
$s_{1,334}$	0.192
$s_{1,2}$	0

Table 5: Approximate KNN similarities for 10 partitions and 2 replications, $k = 10$.

AK.2 Vary the number of partitions in which a given user appears. For the `data/ml-100k/u2.base` training set, partitioned equally between 10 workers, report the relationship between the level of replication (1,2,3,4,6,8) and the MAE you obtain on the `data/ml-100k/u2.test` test set. What is the minimum level of replication such that the MAE is still lower than the baseline predictor of Milestone 1 (MAE of 0.7604), when using $k = 300$? Does this reduce the number of similarity computations compared to an exact k-NN? What is the ratio? Answer both questions in your report.

The following Table 6 represents acquired MAE results on 100k dataset with 10 partitions and different level of replication: 1,2,3,4,6,8.

Also see Figure 1 for the visual comparison of Exact KNN MAE, and Approximate KNN MAE for various replicate numbers.

Replication	MAE
1	0.8056
2	0.7548
3	0.7433
4	0.7431
6	0.7410
8	0.7390

Table 6: Relationship between the level of replication (1,2,3,4,6,8) and the MAE for $k = 300$ and 10 workers.

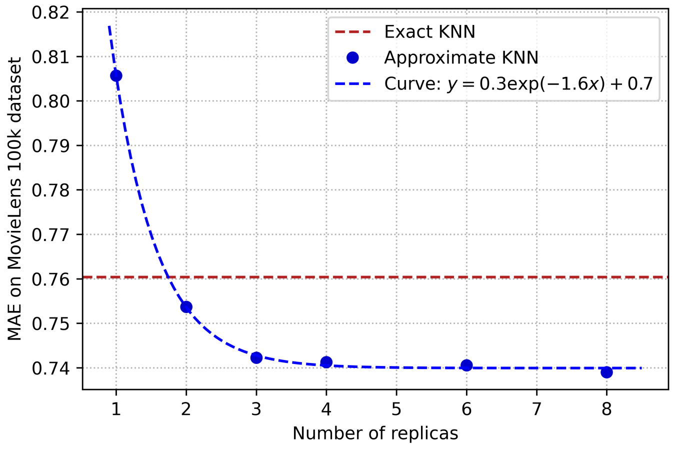

Figure 1: MAE for approximate KNN for various amounts of replication ($k = 300$, 10 partitions, 100k dataset).

The minimum number of replications so that the MAE is lower than in the baseline is 2, so we get: $0.7548 < 0.7604$.

The number of similarity computations is reduced compared to exact-KNN. Mainly it is reduced by making several partitions of data so that we have the following number of approximate similarity computations:

$$approxPartitions = (Users/partitions)^2 * partitions$$

By adding the replicas, we increase the number of similarity computations:

$$approxReplicas = (Replicas * Users/partitions)^2 * partitions$$

Thus the number of computations is reduced compared to the exact-Knn:

$$exact = Users * Users$$

AK.3 Measure and report the time required by your approximate *k-NN* implementation, including both training on `data/ml-1m/rb.train` and computing the MAE on the test set `data/ml-1m/rb.test`, using $k = 300$ on 8 partitions with a replication factor of 1 when using 1, 2, 4 workers. Perform each experiment 3 times and report the average and standard-deviation. Do you observe a speedup compared to the parallel (exact) k-NN with replicated ratings for the same number of workers?

The results are presented in Table 7.

Workers	Avg time, s	Std, s
1	56.3	0.8
2	52.2	0.4
4	50.4	0.8

Table 7: Average time and standard-deviation across 3 runs of MAE computation for the Approximate KNN algorithm using $k = 300$ on 8 partitions with 1 replica (on the IC cluster).

We don't see significant increase in the speed of execution compared to exact-Knn, but that may be caused by the high demand of cluster resources at the time of measuring.

We note that, unlike MPI, Spark does not offer API for Scatter (sending a specific information array to each of the processes, instead of the same one, like Broadcast does). We have not found a way to implement it by hand. Most of our attempts ended up at Spark throwing an exception indicating that the proposed parallelization is not possible. Thus, we still use broadcast in our implementation.

We broadcast an array of "partial" matrices: element i of this array is a matrix of ratings for users belonging to the partition i. The worker then selects the corresponding matrix, and works only with it. This approach can be easily adapted to a Scatter, if a way to do it is found.

The need to broadcast the full ratings array, as well as an additional overhead created by the need to compute partitions, and create partition ratings matrices, does not allow our implementation to show a drastic increase in speed over the simpler parallelization.

We also note that when collecting partial similarity matrices it stands to reason, in the case when two partial matrices yield different similarity for the same pair of users, to take maximum of those similarities. This could happen for example if one similarity. However, this approach is not compatible with our implementation since, to the best of our knowledge, when using a CSCMatrix.Builder, there is no way to access the elements that were already added to the builder. An alternative would be to instead map over all pairs of similarities, obtained from partial matrices. This approach, however, presents a strange mix of both Spark and Breeze approach, and we decide to settle for the initial approach. This way our approximate KNN may have inprecise similarities sometimes, but, on the other hand, it is more easily comparable to the parallel implementation of KNN.

4 Economics

Implement the computations for the different answers in the Economics.scala file. You don't need to provide unit tests for this question, nor written answers for these questions in your report.