Project-2 Report

This report presents an overview on the implementation of a near-neighbour algorithm over Apache Spark as a part of the Project-2 for the Database Systems (CS-422) course at EPFL. The aim of the project is to process input queries using different near-neighbour implementations based on Jaccard similarity on a movie dataset containing a list of keywords.

Different Algorithms

ExactNN

In this naïve near-neighbour implementation, the distance of each query point against all the available data points is calculated using Jaccard similarity. It returns the pairs which pass a threshold value. This is a very computationally expensive process (takes the longest to run) as it involves huge cartesian and grouping operations, but it is highly accurate in terms of precision and recall. It has the lowest average Jaccard distance (1 – similarity) for appropriate parameters.

BaseConstruction

In this implementation based on Locality-Sensitive-Hashing (LSH), the list of keywords is approximated using a MinHash value based on Jaccard similarity. The movies with the same MinHash value are grouped together locally in buckets, which can be easily accessed for matching queries. This saves a lot in terms of computation time, but only offers an approximate result (lower precision and recall).

BaseConstructionBalanced

In this variant of BaseConstruction, a load-balancing mechanism is deployed to reduce the effect of skew. This is achieved by partitioning the queries and buckets into an equi-depth histogram. The histogram partitions are created based on the number of queries using a custom partitioner class. The remaining algorithm is the same as before except it now uses the partitioned variables. This algorithm performs the same as the basic version in terms of precision and recall. Normally, it takes longer to run because of the overhead involved in partitioning. However, the improvement in execution time is visible when larger datasets and skewed queries are involved.

BaseConstructionBroadcast

In this variant of BaseConstruction, a broadcasting scheme is used in order to reduce the query point shuffles. This is possible because the buckets are available globally to every node. The algorithm is the same as before except for the broadcasted buckets. This algorithm performs the same as the basic version in terms of precision and recall, but it is much faster to execute in all scenarios.

ANDConstruction

In this composite implementation, multiple LSH-based functions are ANDed by simply performing an intersection on the set of results. Since the result is a smaller set of results out of all inputs, this algorithm performs better in terms of precision. Hence, construction1 was implemented solely as an AND composition of 4 BaseConstructions with different seeds.

ORConstruction

In this composite implementation, multiple LSH-based functions are ORed by simply performing a union on the set of results. Since the result is a bigger set of results out of all inputs, this algorithm performs better in terms of recall. Hence, construction2 was implemented solely as an OR composition of 5 BaseConstructions with different seeds.

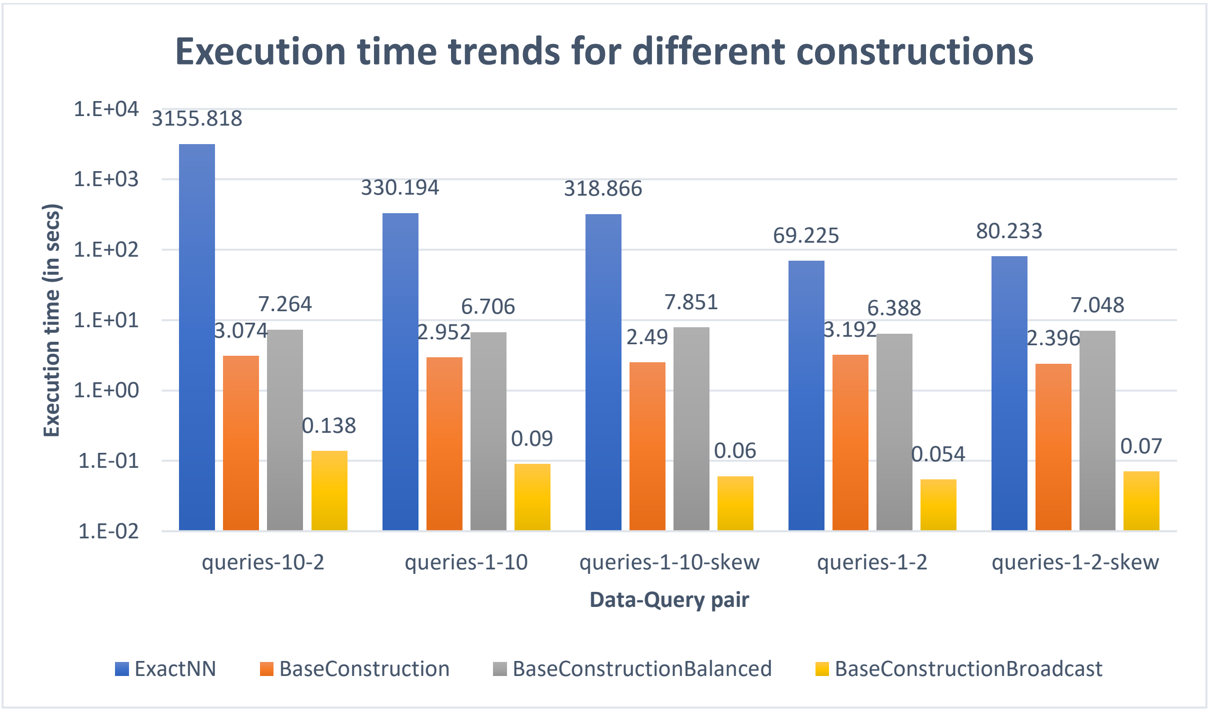
Results

For the final task, a simple algorithm was written to test the 3 BaseConstruction variants against ExactNN as the ground truth. The comparisons were done in terms of execution time (for speed), precision and recall (for accuracy). Further, the average distance of each query point from each nearest neighbour was calculated using time-consuming join operations.

The experiments were performed for different seed values, different thresholds, and different datasets and queries. But only the results for corpus-1 and queries-1-2 are presented here. For the other datasets (1-10, 10-2), average distance calculation was computationally very expensive, and hence, only the other parameters – execution time, precision and recall – were recorded. After multiple attempts on the local machine and the cluster, it was not possible to obtain the results due to various errors – out of Java heap space, and GC overhead limit exceeded (could not be solved even with TA's suggestions). The remaining huge datasets and queries were skipped too.

Execution Time trends for different constructions

We can observe that ExactNN takes the longest, by a huge margin, to execute compared to other implementations. This is due to the expensive cartesian operation on huge datasets. Among the three LSH-based implementations, BaseConstructionBroadcast is the fastest since the buckets are available globally to all the nodes. And lastly, the execution time is much larger for larger datasets. This can be observed from the figure below, plotted with a logarithmic scale for seed = 45 and threshold = 0.3. Similar results are obtained for other variations too.



Skewed Dataset trends

The above figure illustrates the trend for skewed queries in small datasets. BaseConstructionBalanced performs slower than BaseConstruction for skewed and non-skewed queries because of the additional partitioning overhead. However, the advantage of BaseConstructionBalanced can be seen for larger datasets and skewed queries, where it performs faster than the basic version. Unfortunately, it could not be presented in this chart (queries-10-2-skew ran into Java heap space error despite all attempts).

Precision and Recall trends

The following observations about accuracy are consistent with all the seed and threshold values for corpus-1 data and queries (and logically must be true for others as well):

1. ExactNN always performs with a precision and recall of 1.0 since it directly uses the Jaccard similarity without any approximations.
2. The three variants of BaseConstruction have the exact same precision and recall because they use the same MinHash method for approximation.
3. The BaseConstruction variants perform slightly worse than ExactNN, as expected. The precision achieved was 1.0 for seed = 42, but it was around 0.65 for seed = 45. However, recall was always around the high end between 0.90 and 0.99.

Average Distance trends

The trends for the average distance (1 – Jaccard similarity) are dependent on the chosen dataset and the seed value. For corpus-1 data and queries, following can be said:

1. For seed = 42: ExactNN has a slightly higher average distance than the three BaseConstruction variants. However, all the values are around 0.5, and the dataset is quite small, hence, no conclusion can be drawn.
2. For seed = 45: ExactNN has a lower average distance than the three BaseConstruction variants, which is consistent for different threshold values too. The difference here is quite significant, 0.5 for ExactNN vs 0.8 for the other three implementations. This seems logical since the LSH-based implementations rely on approximations, and hence, the query points are *relatively farther* from the nearest data point neighbours, on average.

Conclusion:

The project has been completed successfully and it has passed all the tests. To conclude, ExactNN algorithm is time-expensive but an accurate method. But it is not very practical to use it, given that the LSH-based methods give reasonably accurate performance at much higher speeds. Broadcasting significantly reduces the execution time for smaller datasets. Load balancing partitions the data buckets and queries which offers an advantage with skewed queries. This cannot be seen for the smaller datasets due to the overhead, but it is more apparent in the larger datasets. However, all three variants perform the same in terms of accuracy. In terms of average Jaccard distance (1 – similarity), it can be said that ExactNN has the smallest average distance compared to the other implementations for sufficiently large datasets or appropriate seed values.

Finally, AND and OR compositions of the constructions can be used to further improve the accuracy in the intended way. ANDconstruction can be used to improve the precision, whereas ORconstruction can be used to improve the recall.

Submitted by: Kushagra Shah (316002)

Dated: 19/05/2021