# CS422 - Project 2 - Near Neighbor Algorithm

Liangze Jiang, 321659

May 2021

## Contents



# 1 ExactNN (Task1)

To implement the naive near-neighbor algorithm, we first notice that there are duplicates in the queries, i.e. the same movie can be queried multiple times. Therefore, it is necessary to maintain a unique id for every query, so that when we groupBy the results, the queries with the same movie name won't be grouped together.

To realize this, we first zip a unique id with the query, then perform a cross product between query and data, and find every pair with an above threshold Jaccard similarity:

$$J(I(u), I(v)) = \frac{|I(u) \bigcap I(v)|}{|I(u) \bigcup I(v)|} = \frac{|I(u) \bigcap I(v)|}{|I(u)| + |I(v)| - |I(u) \bigcap I(v)|} \tag{1}$$

where $I(u)$ and $I(v)$ are the set of keywords of query and data pair. We use scala function intersect() and union() to implement it and handle the empty set scenario. Then we group by the previous unique id and get the final results.

## 2 MinHash (Task2)

To implement MinHash computations, we only need to get the list of keywords, given a record. Then, we map list of string keywords into list of integer and find the minimum as the position of this record. This can be done by map() and min().

## 3 BaseConstruction (Task3&4)

To implement BaseConstruction class, we first need to use the MinHash class to separate our corpus into buckets, and each bucket is specialized by the position in MinHash. Then, we also use MinHash class to generate positions for every queries and finally join the queries and the buckets.

This means for a single query with MinHash value x, its neighbors are the items in the buckets with the same MinHash value. Finally, we use left outer join to avoid no match scenario between queries and buckets.

## 4 BaseConstructionBalanced (Task5)

One possible problem in the BaseConstrction is that some buckets are searched more frequently, resulting in a situation that some executor performs more computations than other, i.e stragglers, so it's time to find a way that let all the executors approximately have the same computation load.

First, we build the buckets for corpus and queries as before, then we compute MinHash histogram for queries, i.e. we compute the number of queries in each buckets. In addition, we use this histogram to separate queries buckets into balanced buckets, which could contain several original buckets, and get the boundary of buckets in order to partition the corpus buckets as well. Finally, we join on the balanced buckets of corpus and queries to find neighbors for each query in a balanced way.

## 5 BaseConstructionBroadcast (Task6)

In the BaseConstructionBroadcast class, the building buckets process is the same as BaseConstruction, but now we imagine that the corpus can fit in memory of each executor. So we collect the buckets to a map and then boradcast it to every executor. By this way we can reduce a lot of communication when matching queries and buckets.

## 6 AND&OR Construction (Task7)

It is natural to apply different seed in MinHash class and get different buckets in order to get different neighbor results, and finally combine several results set to get better results. It is like the bagging strategy in data mining and machine learning.

Through the test we can see that construction1 requires higher precision while construction2 requires higher recall. We know that pick the intersection of different LSH results can improve the precision because many are sure about the results, it is more likely to be a real neighbor. Similarly, the union of different LSH results can improve the recall because the quries are likely to have more neighbors, which increases the recall.

Based on the previous discussion, we combine three LSHs in construction1 and four LSHs in construction2. The resulting precision and recall perfectly pass the tests.

# 7 Performance, Accuracy, Time, Average Distance (Task8)

To measure the average distance of each query point from each nearest neighbors, we compute the average of the average of similarity between queries and their neighbors, thus the higher the final result, the lower the average distance and the better the model accuracy.

Then we tested the average similarity, recall, precision and time for each queries on each neighbor algorithm. We use similarity to replace distance here, because distance can be simply regarded as 1 - similarity, and the bigger the similarity, the lower the distance. In addition, We may find that the similarity, recall and precision of three classes BaseConstruction, BaseConstuctionBalanced and BaseConstructionBroadcast will be the same, given the same corpus and query. We think this scenario is valid because all of these three methods are finding the nearest neighbors based on finding the corresponding bucket, they have the same bucket building strategy though different matching strategies for query and its corresponding bucket.

Table 1: The Difference of Average Similarity between ExactNN and Constructions (distances are measured in similarity; bigger similarity, lower distance)

| | ExactNN | Base | Balanced | Broadcast | And | Or |
|---|---|---|---|---|---|---|
| queries-1-2.csv | 0.9446 | 1.0 | 1.0 | 1.0 | 1.0 | 0.7274 |
| queries-1-2-skew.csv | 0.9376 | 1.0 | 1.0 | 1.0 | 1.0 | 0.6818 |
| queries-1-10.csv | 0.9151 | 1.0 | 1.0 | 1.0 | 1.0 | 0.7424 |
| queries-1-10-skew.csv | 0.9344 | 1.0 | 1.0 | 1.0 | 1.0 | 0.6986 |
| queries-10-2.csv | 0.7880 | 0.3529 | 0.3529 | 0.3529 | 0.7887 | 0.2988 |
| queries-10-2-skew.csv | 0.8211 | 0.2942 | 0.2942 | 0.2942 | 0.8047 | 0.2442 |
| queries-10-10.csv | 0.7795 | 0.3600 | 0.3600 | 0.3600 | 0.7833 | 0.3044 |
| queries-10-10-skew.csv | 0.8250 | 0.2942 | 0.2942 | 0.2942 | 0.8005 | 0.2427 |
| queries-20-2.csv | 0.7576 | 0.3190 | 0.3190 | 0.3190 | 0.7465 | 0.2779 |
| queries-20-2-skew.csv | 0.7914 | 0.2571 | 0.2571 | 0.2571 | 0.7646 | 0.2199 |
| queries-20-10.csv | 0.7552 | 0.3197 | 0.3197 | 0.3197 | 0.7408 | 0.2770 |
| queries-20-10-skew.csv | 0.7889 | 0.2590 | 0.2590 | 0.2590 | 0.7585 | 0.2195 |

## 7.1 Average distance analysis

From the table of average similarity (1 - average distance), we can find some laws. First, we can find that the average similarity is closely related to the precision (compare to table 3). Second, combining just two lsh to form a AndConstruction can significantly reduce the gap of average distance between exactNN and BaseConstuction. Also, we can see that for the first several query sets, LSH can reach a higher average similarity than exactNN, I think it is because queries have only one neighbor (i.e. themselves) in the selected bucket while in exactNN we find all true neighbor for a given query. However, when the query set is getting bigger, approximation methods get worse performance because the selected bucket for a query is filled with some fake neighbors.

Table 2: Recall of each query set

| | ExactNN | Base | Balanced | Broadcast | And | Or |
|---|---|---|---|---|---|---|
| queries-1-2.csv | 1 | 0.9139 | 0.9139 | 0.9139 | 0.9139 | 0.9535 |
| queries-1-2-skew.csv | 1 | 0.9501 | 0.9501 | 0.9501 | 0.9501 | 0.9566 |
| queries-1-10.csv | 1 | 0.8597 | 0.8597 | 0.8597 | 0.8597 | 0.9078 |
| queries-1-10-skew.csv | 1 | 0.9670 | 0.9670 | 0.9670 | 0.9670 | 0.9707 |
| queries-10-2.csv | 1 | 0.7900 | 0.7900 | 0.7900 | 0.7318 | 0.8494 |
| queries-10-2-skew.csv | 1 | 0.8378 | 0.8378 | 0.8378 | 0.8068 | 0.8547 |
| queries-10-10.csv | 1 | 0.7801 | 0.7801 | 0.7801 | 0.7167 | 0.8450 |
| queries-10-10-skew.csv | 1 | 0.8831 | 0.8831 | 0.8831 | 0.8528 | 0.8968 |
| queries-20-2.csv | 1 | 0.7690 | 0.7690 | 0.7690 | 0.6965 | 0.8381 |
| queries-20-2-skew.csv | 1 | 0.8469 | 0.8469 | 0.8469 | 0.8039 | 0.8911 |
| queries-20-10.csv | 1 | 0.7719 | 0.7719 | 0.7719 | 0.6947 | 0.8385 |
| queries-20-10-skew.csv | 1 | 0.8611 | 0.8611 | 0.8611 | 0.8187 | 0.9101 |

## 7.2 Recall analysis

From the table of recall, when we check the recall vertically (between the query sets), we would see that for the same corpus, querying skew set will result in a higher recall compared to balanced set. In addition, when we check the recall horizontally (between the methods), we can see that OrConstruction will reach the highest recall and AndConstruction will have a lower recall. It is because OrConstruction retrieve as much neighbors as it can (as long as the neighbor has shown up in the results of a LSH). Therefore, we get more neighbors retrieved and naturally higher recall.

## 7.3 Precision analysis

Compared to the recall table, the analysis of precision table are totally opposite. We would see that for the same corpus, querying skew set will result in a lower precision compared to balanced set, we can see that AndConstruction will reach the highest precision and OrConstruction will have a lower precision than others. This is easy to understand: AndConstruction pick the neighbors in every LSH, so it is likely that the neighbors are indeed the true positive (because every LSH regards them as true positive); however, the OrConstruction says that we should retrieve all the neighbors that shows up at least once, so it is more likely to make a mistake.

Table 3: Precision of each query set

| | ExactNN | Base | Balanced | Broadcast | And | Or |
|---|---|---|---|---|---|---|
| queries-1-2.csv | 1 | 1.0 | 1.0 | 1.0 | 1.0 | 0.7137 |
| queries-1-2-skew.csv | 1 | 1.0 | 1.0 | 1.0 | 1.0 | 0.5176 |
| queries-1-10.csv | 1 | 1.0 | 1.0 | 1.0 | 1.0 | 0.7380 |
| queries-1-10-skew.csv | 1 | 1.0 | 1.0 | 1.0 | 1.0 | 0.5180 |
| queries-10-2.csv | 1 | 0.3767 | 0.3767 | 0.3767 | 0.8278 | 0.3110 |
| queries-10-2-skew.csv | 1 | 0.1480 | 0.1480 | 0.1480 | 0.9118 | 0.1009 |
| queries-10-10.csv | 1 | 0.3871 | 0.3871 | 0.3871 | 0.8236 | 0.3217 |
| queries-10-10-skew.csv | 1 | 0.1419 | 0.1419 | 0.1419 | 0.9153 | 0.0956 |
| queries-20-2.csv | 1 | 0.3362 | 0.3362 | 0.3362 | 0.7859 | 0.2869 |
| queries-20-2-skew.csv | 1 | 0.1058 | 0.1058 | 0.1058 | 0.8652 | 0.0844 |
| queries-20-10.csv | 1 | 0.3438 | 0.3438 | 0.3438 | 0.7856 | 0.2924 |
| queries-20-10-skew.csv | 1 | 0.0935 | 0.0935 | 0.0935 | 0.8718 | 0.0714 |

Why precisions are so poor on the tests of BaseConstruction, BaseConstructionBalanced and BaseBroadcast on some query set? First, we can think that these three methods are based on bucket matching strategy, and bucket is decided by minimum hash value of keywords list, but minimum hash value does not guarantee the movie as a neighbor (it's an approximation), so it's natural that when corpus is getting bigger, more buckets are filled with more fake neighbors and when our query matches one of these buckets, the precision won't be very high.

Table 4: Time measurement of each query set

| | ExactNN | Base | Balanced | Broadcast | And | Or |
|---|---|---|---|---|---|---|
| queries-1-2.csv | 1.3579 | 0.0706 | 0.0917 | 0.0188 | 0.0492 | 0.0382 |
| queries-1-2-skew.csv | 1.6602 | 0.0843 | 0.1517 | 0.0383 | 0.0556 | 0.0442 |
| queries-1-10.csv | 5.6278 | 0.1023 | 0.1325 | 0.0514 | 0.0900 | 0.0628 |
| queries-1-10-skew.csv | 6.0003 | 0.1036 | 0.1308 | 0.0525 | 0.0739 | 0.0650 |
| queries-10-2.csv | 104.7206 | 0.1397 | 0.2988 | 0.1000 | 0.3020 | 0.3155 |
| queries-10-2-skew.csv | 95.5734 | 0.1688 | 0.2798 | 0.0706 | 0.2531 | 0.2528 |
| queries-10-10.csv | 504.9006 | 0.3425 | 0.6244 | 0.2363 | 0.5565 | 0.5078 |
| queries-10-10-skew.csv | 403.2320 | 0.2671 | 0.3937 | 0.2359 | 0.3987 | 0.4060 |
| queries-20-2.csv | 377.7209 | 0.2210 | 0.4347 | 0.1680 | 0.6683 | 0.5803 |
| queries-20-2-skew.csv | 289.3342 | 0.1929 | 0.3678 | 0.0986 | 0.4868 | 0.4783 |
| queries-20-10.csv | 1776.9492 | 0.5725 | 0.9161 | 0.4204 | 1.5211 | 1.3139 |
| queries-20-10-skew.csv | 1315.8443 | 0.5674 | 0.7830 | 0.4177 | 0.8591 | 1.0962 |

## 7.4 Time measurements analysis

To better visualize the time difference between different methods, we plot the table as bar plots. We can see that exactNN requires a lot more time than other methods, and the time increases very fast when we test on bigger corpus and query, it is because the implementation of exactNN involves

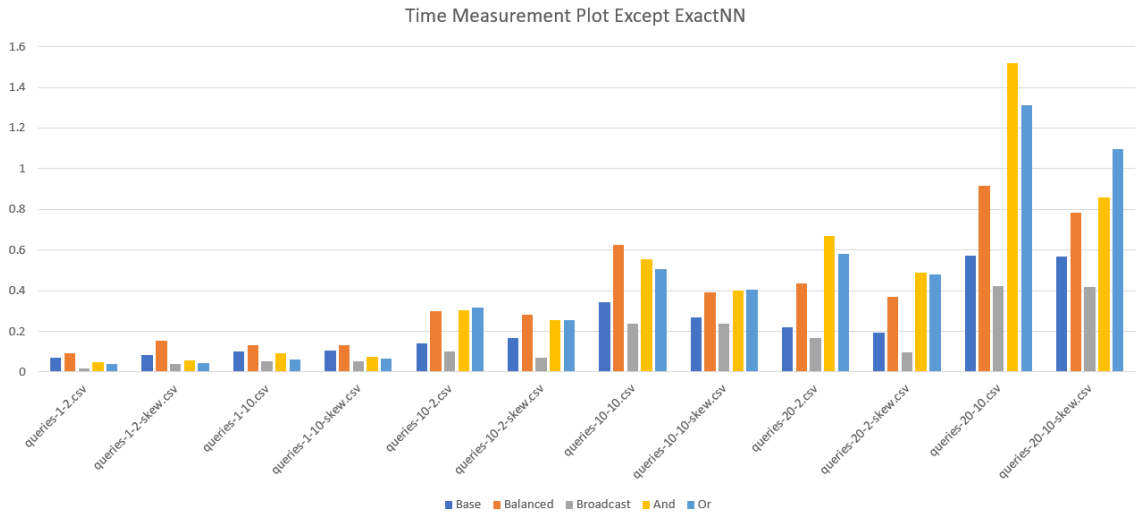


Figure 1: Time Measurement without exactNN

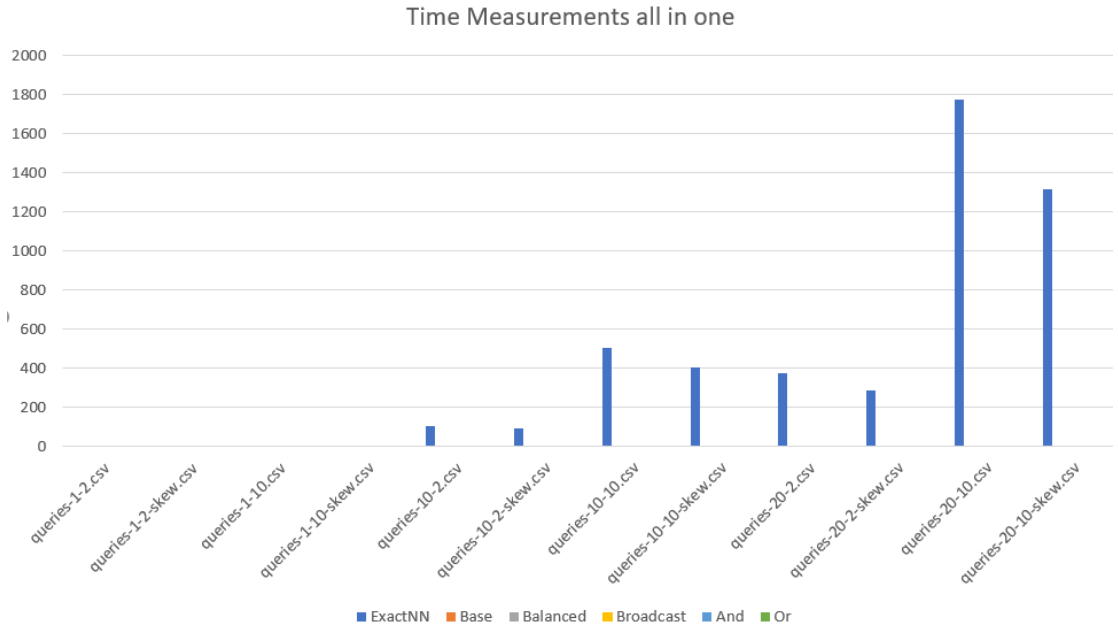


Figure 2: Time Measurement with all methods

a cross product procedure, whose computations grow very fast when the left and right operator get bigger.

Compare the time between approximate solutions, we can find that the broadcast method always

takes less time than others, then the BaseConstruction takes second less time. For balanced, AND and OR methods, balanced methods requires to compute histogram and partition boundary and then form the partitions, and AND, OR methods are the combination of several LSHs, which requires more computation. Hence, they take more time to finish. Also, these methods' time are growing with dataset getting bigger, but growing much slower than exactNN. Last but not least, by checking the execution time of BseConstructionBalanced, we can see that it takes less time on skewed set than non-skew set maybe due to the balanced strategy.

### 7.5 When is each method preferable?

To summarize the performance and accuracy as well as time measurements and average similarity(1-distance):

- For small data set, we don't have to hesitate on which method to choose, just choose one of the method except exactNN, because they don't vary much on time and result in a good accuracy compare to exactNN.
- We can see that, under these settings, broadcast method is less time-consuming and preferable when we are in a time restriction situation. When the query set is very skew, the overhead of balanced method may help it in the end, so we can choose it in that case.
- When we hope to get better results, we should go for AndConstruction and OrConstruction and also other composite methods (which are not covered in our tests).
- Specifically, intersecting several LSHs will lead to more precise result, and this is more robust; combining several LSHs will result in higher recall, and by this we can retrieve more possible neighbors.
- To guarantee both time performance and accuracy, I would go for composition methods of several broadcast LSHs.