K-Maximum Inner Product Neighbor Graph for Inner Product Search

BAO Ergute
CUHK
1155076717@link.cuhk.edu.hk

James CHENG
CUHK
jcheng@cse.cuhk.edu.hk

1. ABSTRACT

We consider the problem of designing a graph based algorithm for maximum inner product search. We first review the L2-ALSH method proposed Shrivastava and Li (2014a). Then we review the SIMPLE LSH and SIMPLE ALSH method prposed by Neyshabur and Srebro (2015), both of which is state-of-the-art. Then we present our K-MIP Graph method. At last we compare the performances of K-MIP Graph with previously mentioned LSH methods on real world data sets.

Keywords
Inner Product; Similarity Search; LSH; K-Nearest Neighbor Graph.

2. INTRODUCTION

We consider the problem of Maximum Inner Product Search (MIPS): given a collection of "database" vectos $S \subset R^d$ and a query $q \in R^d$, find a data vector maximizing the inner product with the query:

$$p = \mathbf{argmax}_{x \in S} q^T x \qquad (2.1)$$

MIPS problems of the form (2.1) arise, e.g. when using matrix-factorization based recommendation systems, structral SVM and Drop Out in Artificial Neural Networks, etc.

It is worthy to mention that MIPS is closely related to Nearest Neighbor Search (NNS):

$$p = \mathbf{argmin}_{x \in S} \|q - x\|_2 \qquad (2.2)$$

In the case that $\|x\|_2 = c$ for all $x \in S$ and c being a constant, then MIPS and NNS are equivalent, although mostly that's not the case. But it is clear that both (2.1) and (2.2) can be categorized as Similarity Search problems with different metrics for 'similarity'. In the literature, many methods have been proposed. They include data independent methods, the most famous one of which being LSH (Indyk, 2004), data dependent methods, e.g. ITQ (Gong and Lazebnik, 2011), product quantization methods, e.g. OPQ (Ge et al,2015) and graph based methods, e.g. KNN Graph (Dong et al, 2011) and HNSG (Malkov and Yashunin).

However, in the literature to find exact or approxiamte MIPS solutions was seldom studied until recently. Tree-based methods have been prposed for MIPS, e.g. MIPS Using Cone Trees (Ram and Gray, 2012). However Shrivastava and Li (2014a) argued that tree-based methods are impractical in high dimensions while the LSH-based methods have good performance guarantees regarless of the dimension of the data. Shrivastava and Li proposed L2-ALSH (2014a) to solve MIPS based on LSH. Later Neyshabur and Srebro (2015) proposed SIMPLE LSH and SIMPLE ALSH with better theoretical guarantees and improved empirical results.

Our contribution is a proposing a graph based algorithm called K-Maximum Inner Product Neighbor Graph (K-MIP Graph) for solving MIPS problem with roughy the same empirical results on some data sets compared with SIMPLE LSH/ALSH, which is state-of-the-art. K-MIP Graph method is purely heuristic with no theoretical gurantees. K-MIP Graph has its own applied area, e.g. small world graph, recommendation systems, etc. The idea of K-MIP Graph comes from K-Nearest Neighbor Graph (KNN Graph) (Dong et al, 2011) which was originally used for the NNS problem.

A K-MIP Graph is a graph like KNN Graph but using a diffenrent similarity metric. Unlike in KNN Graph, where we expand each node according to the Euclidean distance to the query, here in K-MIP Graph we expand according to the inner product with the query. Emprical results show that K-MIP Graph roughly has the same performance with SIMPLE LSH and SIMPLE ALSH in some data sets.

The rest of this report will be arranged as the following: In section 3 we will first review L2-ALSH (Shrivastava and Li, 2014a) and SIMPLE LSH/ALSH (Neyshabur and Srebro, 2015) and our experimental results on real world data sets. Then in section 4 we will discuss K-MIP Graph method in detail show its performance. In section 5, we will compare K-MIP with SIMPLE LSH/ALSH. Lastly, in section 6 we will discuss the future work.

3.0 Maximum Inner Product Search by LSH

In this section, we will have a review on how to find the maximum inner product by using LSH and ALSH.

From our review on Similarity Seach problems, we found that NNS is heavily studied whereas MIPS is rarely studied. As we've mentioned before, (2.1) and (2.2) are equivalent in the case that $\|x\|_2 = c$ for all $x \in S$ and c being a constant. If there is some mechineries to convert (2.1) to (2.2), then we can use the algorithms for NNS to solve MIPS easily. Shrivastava and Li (2014a) first present the method of Asymmetric Hashing to convert (2.1) into (2.2) and then use LSH to solve the problem. Following Shrivastava and Li, Neyshabur and Srebro (2015) come up with SIMPLE LSH/ALSH and basically solve the MIPS problem for good and don't leave much else for us to discover. Because SIMPLE LSH/ALSH have both strong theoretical gurantees and better emprical performance than other heuristic algorithms. In the next subsections, we will review L2-ALSH (Shrivastava and Li, 2014a) and SIMPLE LSH/ALSH (Neyshabur and Srebro, 2015) and show our experimental results of SIMPLE LSH/ALSH on real world data sets.

Here we'd better revisit our problem definition:

Given a collection of "database" vectos $S \subset R^d$ and a query $q \in R^d$, find a data vector maximizing the inner product with the query:

$p = \mathbf{argmax}_{x \in S} q^T x$

3.1 L2-ALSH

Shrivastava and Li(2014) were the first to come up with a LSH framework to solve the inner product search problem:

For an integer parameter m and real valued parameters $0 < U < 1$ and $r > 0$, consider the following pair of mappings:

$P(x) = [Ux; \|Ux\|_2^2; \|Ux\|_2^4; \dots; \|Ux\|_2^{2^m}]$ (3.1)

$$Q(q) = [y; 1/2; 1/2; \ldots; 1/2] \quad (3.2)$$

combined with the standard L_2 hash function

$$h_{a,b}^{L_2} = \lfloor \frac{a^T x + b}{r} \rfloor \quad (3.3)$$

where $a \sim N(0,I)$ is a spherical multi-Gaussian random vector, $b \sim U(0,r)$ is a uniformly distributed random variable. The intermediate space after this step of assymetric hashing is R^{d+m} since (3.1) and (3.2) both append m scalars after the original query and data vectors. Note that after this step of assymetric hashing, we have the following three key equalities

$$\|P(x)\|_2^2 = \|x\|_2^2 + \|x\|_2^4 + \ldots + \|x\|_2^{2^m} + \|x\|_2^{2^{m+1}}$$

$$\|Q(q)\|_2^2 = \|q\|_2^2 + m/4 = 1 + m/4$$

$$\|Q(q) - P(x)\|_2^2 = (1 + m/4) - 2q^T x + \|x\|_2^{2^{m+1}} \quad (3.4)$$

where without loss of generality we have assumed that $\|q\|_2 = 1$ and $\|x\|_2 < 1$ by scaling with $0 < U < 1$, i.e. queries of unit length and data points are shorter than unit length. Then we look at (3.4) and we can find that with m tends to infinity and fixed, the problem MIPS (2.1) is equivalent with the problem of NNS (2.2) on the intermediate space . And we can later use any LSH framework, e.g. (3.3), to solve the converted problem. The problem with L2-ALSH (Shrivastava and Li, 2014a) is that neither theoreticaly nor empirically we can have that m tends to infinity and thus it only gives an approximate solution with weaker theoretical guarantee than applying LSH directly in the appropriate vector space. To get rid of m, Neyshabur and Srebro introduces SIMPLE LSH/ALSH (2015) and made the theory more compelete. Nevertheless, Shrivastava and Li' s work should be regarded highly since they first come up with the machinery to convert a MIPS problem to aNNS problem.

3.2 SIMPLE LSH

Following Shrivastava and Li, Neyshabur and Srebro (2015) come up with SIMPLE LSH/ALSH, which is more theoretically compact.

Instead of introducing m, they consider the mapping:

$$P(x) = [x; \sqrt{1 - \|x\|_2^2}] \quad (3.5)$$

In particular, if without loss of generality we have assumed that $\|q\|_2 = 1$ and $\|x\|_2 < 1$ by scaling, we have:

$$P(q) = [q; 0] \quad (3.6)$$

The intermediate space after this step of assymetric hashing is R^{d+1} since (3.5) and (3.6) both append 1 scalar after the original query and data vectors. In addition, both $\|P(x)\|_2 = 1$ and $\|P(q)\|_2 = 1$ for any data point x and query point q. We also have the following key equality:

$$P(q)^T P(x) = q^T x \quad (3.7)$$

since $P(q)$ only gives q a trailing 0. Now that we have (3.6) and $\|P(x)\|_2 = 1$ and $\|P(q)\|_2 = 1$, we can apply sign hash function on the intermediate space R^{d+1} by generating $a \sim N(0,I)$ on the unit sphere. We define the mapping from R^{d+1} to a binary alphabet {1,-1} as:

$$h_a(P(x)) = \mathbf{sign}(a^T P(x)) \tag{3.8}$$

and we have (Goemans and Wiliamson, 1995 Improved approxiamtion algorithms for maximum cut and satisfiability problems using semidefinite programming, Journal of the ACM(JACM). 42:6:111501145):

If $q^T x \geq S$, then

$$Pr[h_a(P(q)) = h_a(P(x))] \geq 1 - \frac{\mathbf{cos}^{-1}(S)}{\pi}$$

If $q^T x \leq cS$, then

$$Pr[h_a(P(q)) = h_a(P(x))] \leq 1 - \frac{\mathbf{cos}^{-1}(cS)}{\pi}$$

Since for any $0 \leq x \leq 1$, $1 - \frac{\mathbf{cos}^{-1}(x)}{\pi}$ is a monotically increasing function, this gives us an LSH.

3.3 SIMPLE ALSH

To make the thoery more compact, Neyshabur and Srebro (2015) also come up with ALSH, which solves a MIPS problem without the assumption that $\|q\|_2 = 1$. The idea were vey similar to SIMPLE LSH, yet this time we have the following assymetric mappings:

$$P(x) = [x; \sqrt{1 - \|x\|_2^2}; 0] \tag{3.9}$$

$$Q(q) = [q; 0; \sqrt{1 - \|q\|_2^2}] \tag{3.10}$$

The very same sign hash function can later be applied on the intermediate space R^{d+2} space and at last, we have an LSH.

3.4 Experiments on SIMPLE LSH/ALSH

We show the experimental result on the two following collaborative filtering datasets, Netflix and Moveliens 10M.

For a given user-item matrix Z, we followed the pureSVD procedure suggested by Cremonesi et al. (2010): we first subtracted the overall average rating from each inidivual rating and created the matrix Z with these average-subtracted ratings for observed entries and zeros for unobserved entries. We then take a rank-f approximation by keeping the top f singular components. $f = 150$ for Movielens and $f = 300$ for Netflix. $Z \approx W \Sigma V^T = Y$ and define $U = W\Sigma$ so that $UV^T = Y$. We can think of each row of U as the vector presentation of a user and each row of V as the presentation of an item.

Here we show the precision-recall curve for the top k maximum inner product neighbor (item) for a given query (user) with $k \in \{1,5,10\}$ and K is the number of hash functions we use, i.e. the number of a in sign hash function we generate.

A trivial note is that SIMPLE LSH/ALSH have diffenrent results, this is mainly due to the precision in my computer. We can tell from the resulting figure that as K gets bigger the result gets better. This is one charcteristic of LSH since larger K gives a tighter bound on the probability. In practice larger K means more computation and thus leads to longer querying time. Last but not the least, we get worse result as k gets smaller and the result for exact maximum inner product is not very good. The above two points I've

mentioned motivates us to come up with our K-MIP Graph method aiming to get shorter querying time and better performance for all k in some real world data sets.

4.0 A New Method: K-MIP Graph

In this section, we introduce a new method called K-MIP Graph for solving MIPS problem. The basic ideas of our method come from K-Nearest Neighbor Graph (KNN Graph) (Dong et al, 2011).

4.1 Implementation: K-MIP Graph

Like on KNN Graph, searching on a K-MIP Graph is simpy greedy. We first preprocess the dataset to build a K-MIP graph, where we store a list of points selected the dataset that compose the top K maximum innerproduct neighbor for each point in the data set. The algorithm starts with a random (or a navigating node fixed before time) or coarsely selected node. At each step, it checks the MIP neighbors of the current node and moves to closer nodes to the query. Originally we have two implementations for "closer", one is the angular distance from nodes to the query, the other is the inner product from nodes to the query. After experimenting we found that the latter one is better. The algorithm repeats this process until on closer nodes can be found. Please see Algorithm.1 for the pseudo-code for the implementation.

Algorithm 1. Search-on-KMIP (G, p, q, h, l, r)

Require: graph G (the K-MIP graph for the item dataset), start node p, query point q, number of epochs for greedy expansion h, candidate pool size l, expansion pool size r.

Ensure: the top k MIP neighbors of q

1.$i = 0$

2.candidate pool $S = \{\}$

3.explore pool $C = \{p\}$

4.while $i < h$ do:

4. $j =$ the index of the first unexpanded node in C

5. mark p_j as expanded and remove p_j from C

6. for all neighbor a of p_j in G do

7. S.add(a)

8. C.add(a)

9. sort C, S according to the inner product w/ q

10. remove the distant nodes in C to keep its size no larger than r

11. remove the distant nodes in S to keep its size no larger than l

12. return the first k nodes in S

Here we can simply pick an abitrary node p as the starting node, our experiments show that the long term behavior of time-recall curve doesn't depend highly on the start node.

4.2 Experiments

We show the experimental result on the two following collaborative filtering datasets, Netflix and Moveliens 10M, the same as in section 3.4.

Here for K-MIP graph, we have $K = \{10,20,50,100\}$ and we showthe following indicators of performance:

1. Number of inner product computed

2. Recall

3. Depth of searching

Here we show the depth-recall curve for the top k maximum inner product neighbor (item) for a given query (user) with $k \in \{1,10\}$ and K is the number MIP neighbors for each data point within the dataset. We expand one node at each step and examine all its K neighbors' inner product w.r.t the query point until a fixed number of steps (depth), here we fix the maximum number of depth to be 100.

Note that for data of dimensions 150 or 300 (which we get from Movielens 10M and Netflix) or even higher, most of the computation during querying occurs when compute the inner product between the query and the data point (extra computation for the hash function in LSH framework).

4.3 Discussion on Empirical Performance of K-MIP

In this subsection we will discuss the experimental results of K-MIP graph on the Movielens 10M and Netflix datasets.

As shown in figure 5 and figure 6, the depth-recall curve is concave. Also, as we can see from figure 3 and figure 4, the first very small proportion of the curve is concave but after that the curve is almost linear. So the deeper we explore the graph, the fewer true MIP neighbor we can get from the same computation cost. For large K, e.g. 100, we may stop the algorithm within 10 to 20 steps if all we want is some approximate result. For small K, e.g. 10 and 20 we may go as deep as 50 or 60 steps to get comparably good result compared with large K.
Originally we only implemented a naiive version, where on each step we forget about which nodes we have expanded, and the recall stops to increase after only 5 to 10 steps. To improve this situation we come to our current implementation, where we keep a list of nodes to be explored of a fixed size r , e.g. 100 (there is no significant difference on recall between the two implementations until we reach a dead end). And we can see from figure 5 and figure 6, the recall will eventually goes to 1 and thus there is no dead end while searching on the K-MIP graph on Movielens 10M and Netflix datasets.

5.1 Comparison with SIMPLE ALSH/LSH

As mentioned in section 4.2, we keep a list of nodes to be explored of a fixed size r while searching on the K-MIP graph. If we implement the list by a binary heap then operations on it will be $O(\log r)$, which is very small compared with the dimension of the data (if the dimension is low, then a tree structure can be used for searching instead of LSH). So here when we compare our algorithm with SIMPLE ALSH/ LSH, 'time' is replaced by 'the number of inner product computed'. For Movielens 10M and Netflix, the dimension of data points is 150 and 300 respectively and we pick r from $\{10,20,50,100\}$, corresponding to K in $\{10,20,50,100\}$ for our K-MIP graph construction.

As we can see from figure 7 and figure 8, compared with Simple LSH/ALSH with small number of hash functions, our algorithm has superior performance as the curve of K-MIP is higher than those of LSH.

Meanwhile when compared with Simple LSH/ALSH with large number of hash functions, our algorithm also has comparable performance as the curve of K-MIP is to the left of the curves of LSH. The reason for this comes from two parts. First, in order to get high recall for a LSH framework, we need a large number of hash functions and when a query comes we need to first apply the same hash functions to the query, which is basically a very large computation cost. Second, if we take only a small number of hash functions, then due to the monotonicity in probability, the precision won't be high. So as we go through hash buckets, the recall will increase very slowly.

Nevertheless, K-MIP graph gives slightly better performance on Movielens 10M and Netflix datasets doesn't justify K-MIP is superior to SIMPLE LSH/ALSH in all aspects and applications. First of all, LSH frameworks have theoretical guarantees:

If $q^T x \geq S$, then

$$Pr[h_a(P(q)) = h_a(P(x))] \geq 1 - \frac{\mathbf{cos}^{-1}(S)}{\pi}$$

If $q^T x \leq cS$, then

$$Pr[h_a(P(q)) = h_a(P(x))] \leq 1 - \frac{\mathbf{cos}^{-1}(cS)}{\pi}$$

Second, LSH frameworks is free of the curse of dimensionality unlike graph structures. Third, to build a K-MIP graph on a large number of data points with high dimensions could be very time consuming while LSH can also be done real time. We can think of the benefits of searching on K-MIP graph come from the cost of construction of the K-MIP graph.

5.2 Space Complexity and Offline Preprocessing

For a K-MIP graph, we keep an adjacency list for every data points in the dataset. For example, if the total number of data point is n, then in total we need extra $O(nK \log n)$ bits. For SIMPLE LSH/ALSH, we only need $O(nK)$ bits, where K is the number of hash functions. But we also need to store the K by d hash matrix. So in terms of space complexity, SIMPLE LSH/ALSH has a noticable advantage.

Moreover, in order to build a K-MIP graph we need to have all data points in advance. And the time complexity to build a exact K-MIP graph is $O(n^2 d \log n)$. When a new data point comes to the dataset, we need to update all existing adjacency list and build one for the new comer, which gives $O(nd \log n)$. But for SIMPLE LSH/ALSH, we can directly apply the hash function to the new comer and update the hash buckets, which gives $O(dK)$ and it is independent of n.

6 Future work

There are much improvement that can be made for constructing a K-MIP graph efficiently and conducting a search on the graph. For construction, one may use parallel computing for datasets too big to fit in one computer. Moreover, since the adjacency lists are independent with each other, parallel computating enables us to construct adjacency lists simultaneously. Here the trade-off is between communication cost and time.

Cong et al. (2018) proposed Fast Approximate Nearest Neighbor Search With The Nagivating Spreading-out Graph, where they argued that an exact KNN graph is not necessary to get good results in NNS. They've also come up with a solution to disconnectivity in real world graphs. Malkov and Yashunin also come up with a graph structure called Hierarchical Navigable Small World Graphs, where they uses a

hierarchical space decomposition to solve the problem of disconnectivity and improved the performance in seraching on small world graphs. The same ideas can also be used for constructing and searching on K-MIP graphs.

Figure 1. Recall-Precision curve of Simple LSH/ALSH on Movielens 10M dataset.

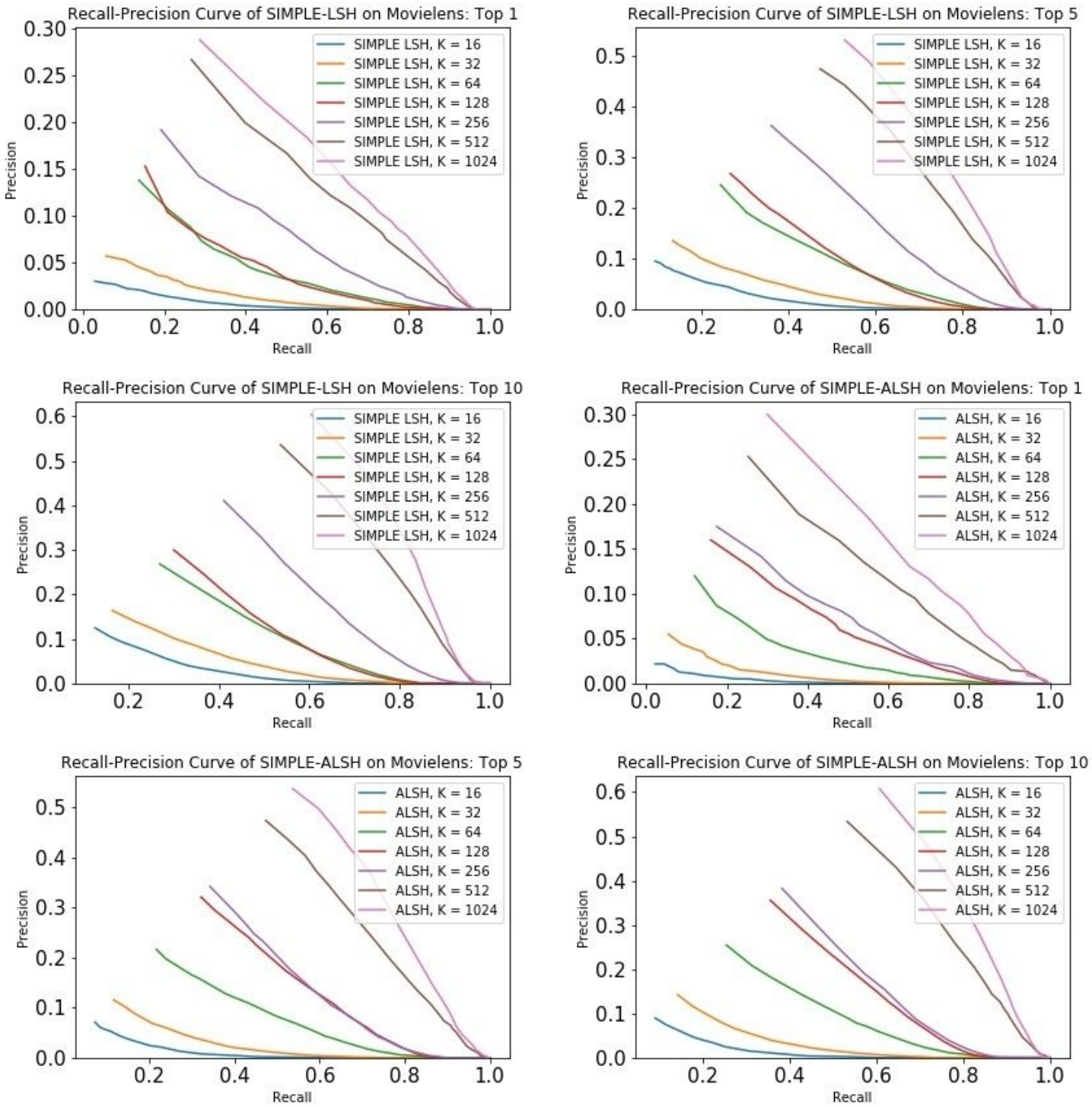
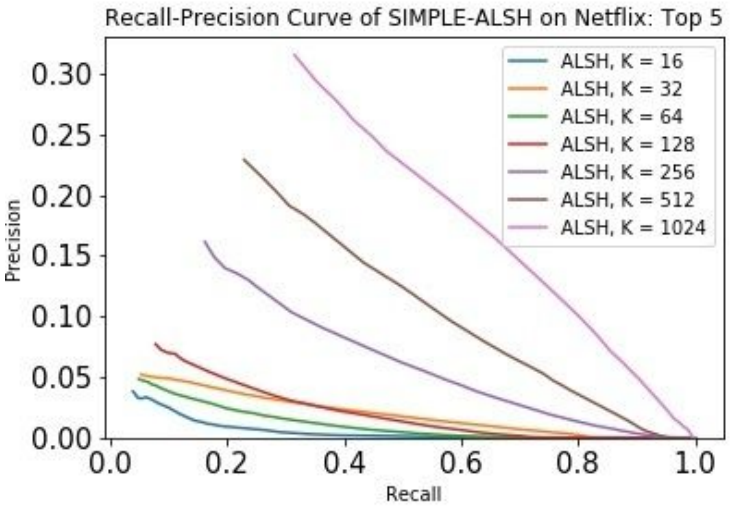
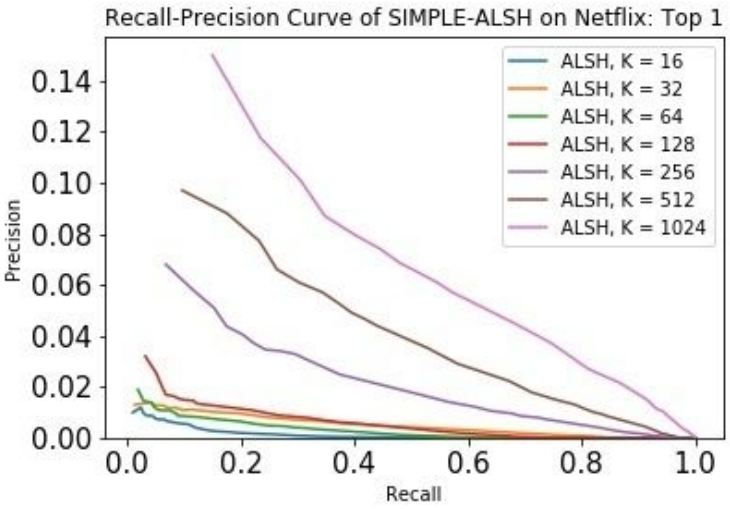
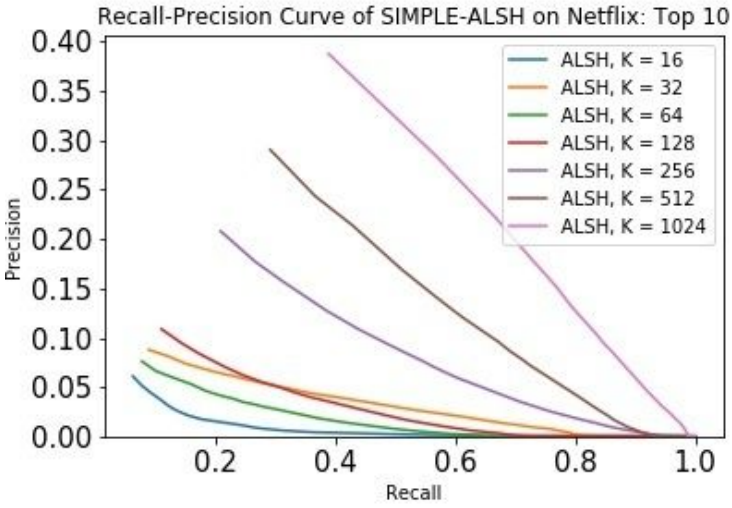
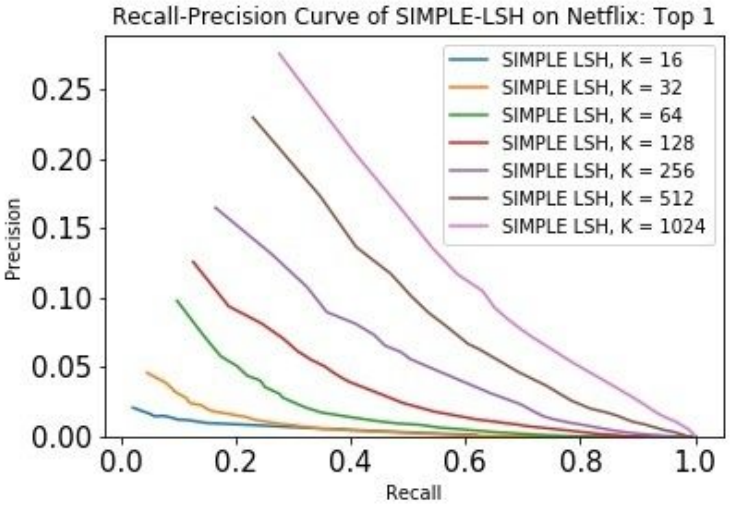


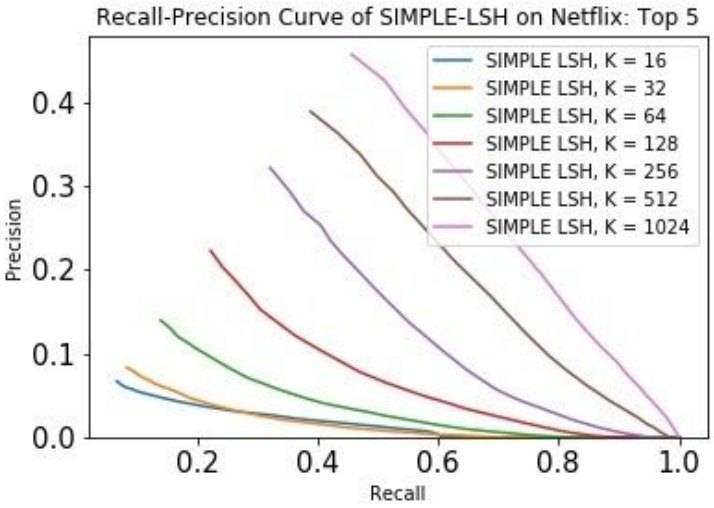
Figure 2. Recall-Precision curve of Simple LSH/ALSH on Netflix dataset.











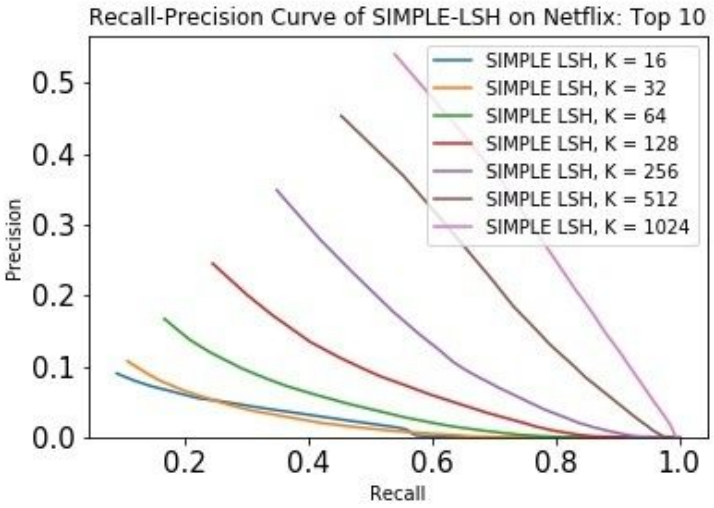


Figure 3. Depth-Computation curve of K-MIP on Movielens 10M dataset.

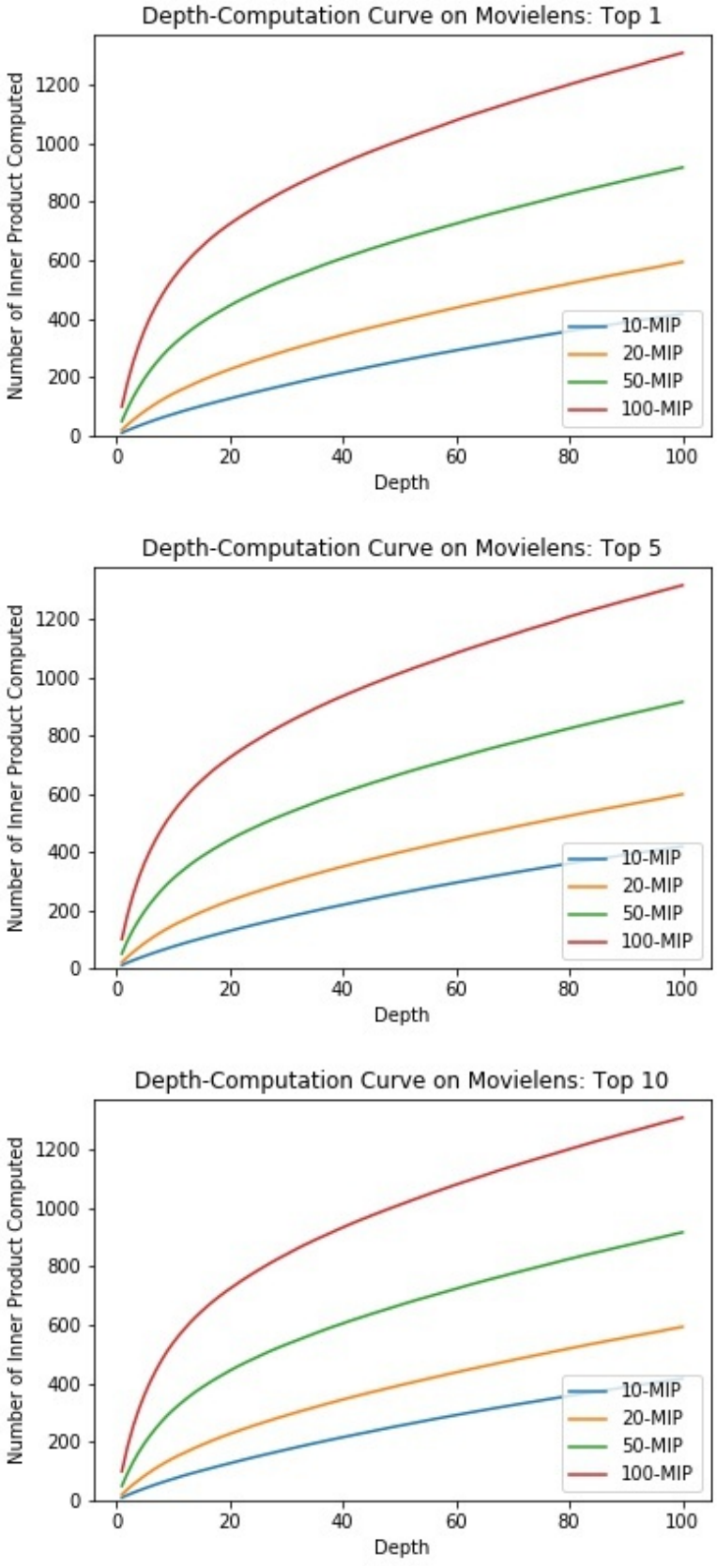


Figure 4. Depth-Computation curve of K-MIP on Netflix dataset.

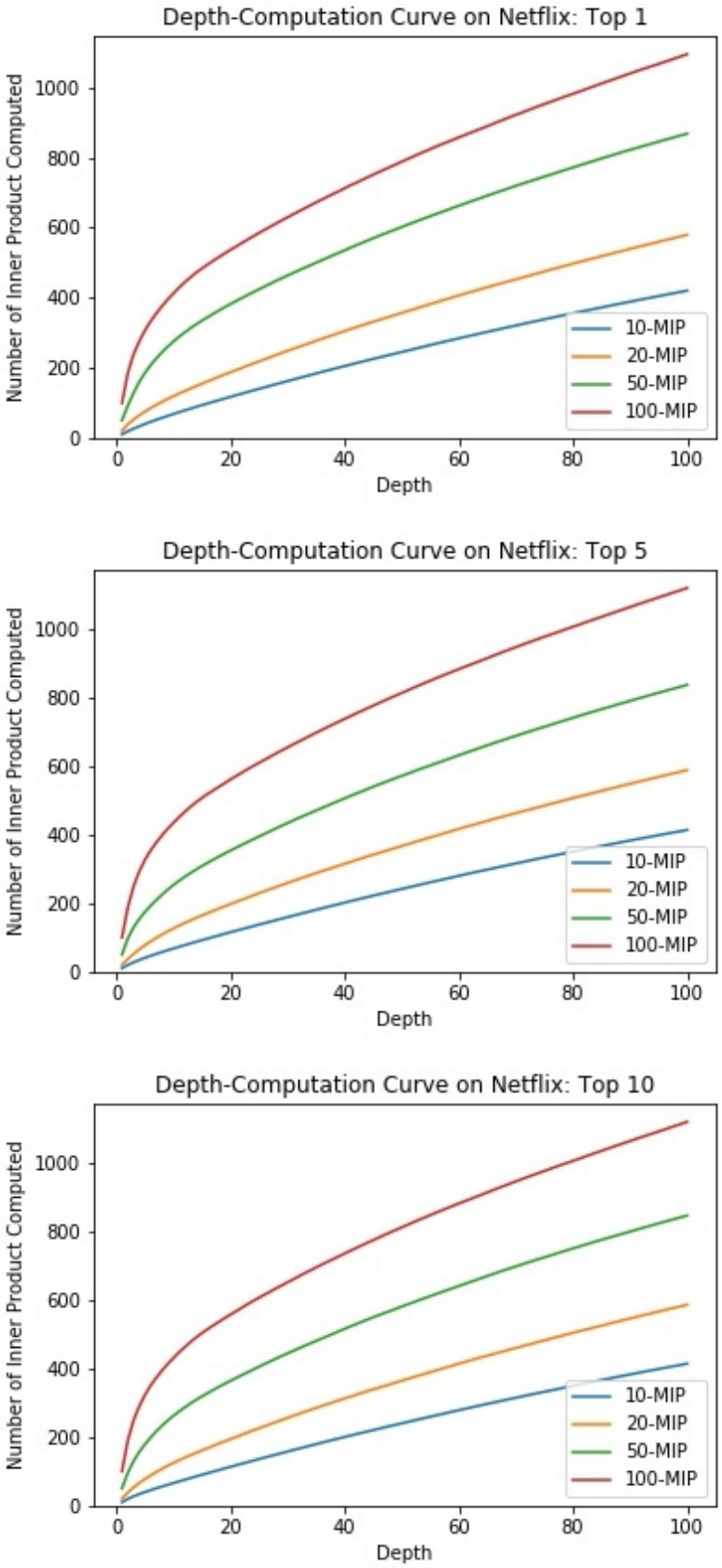


Figure 5. Depth-Recall curve of K-MIP on Movielens 10M dataset.

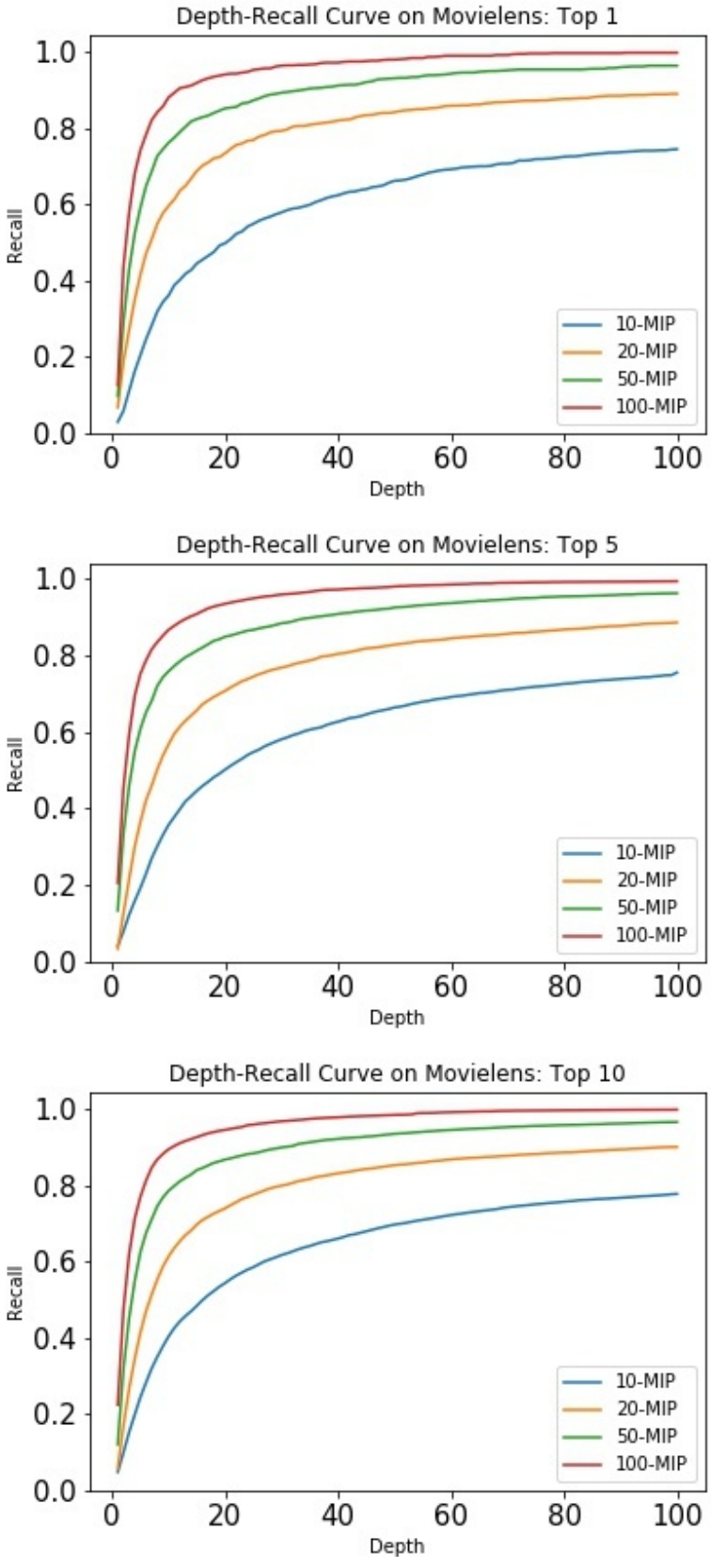


Figure 6. Depth-Recall curve of K-MIP on Netflix dataset.

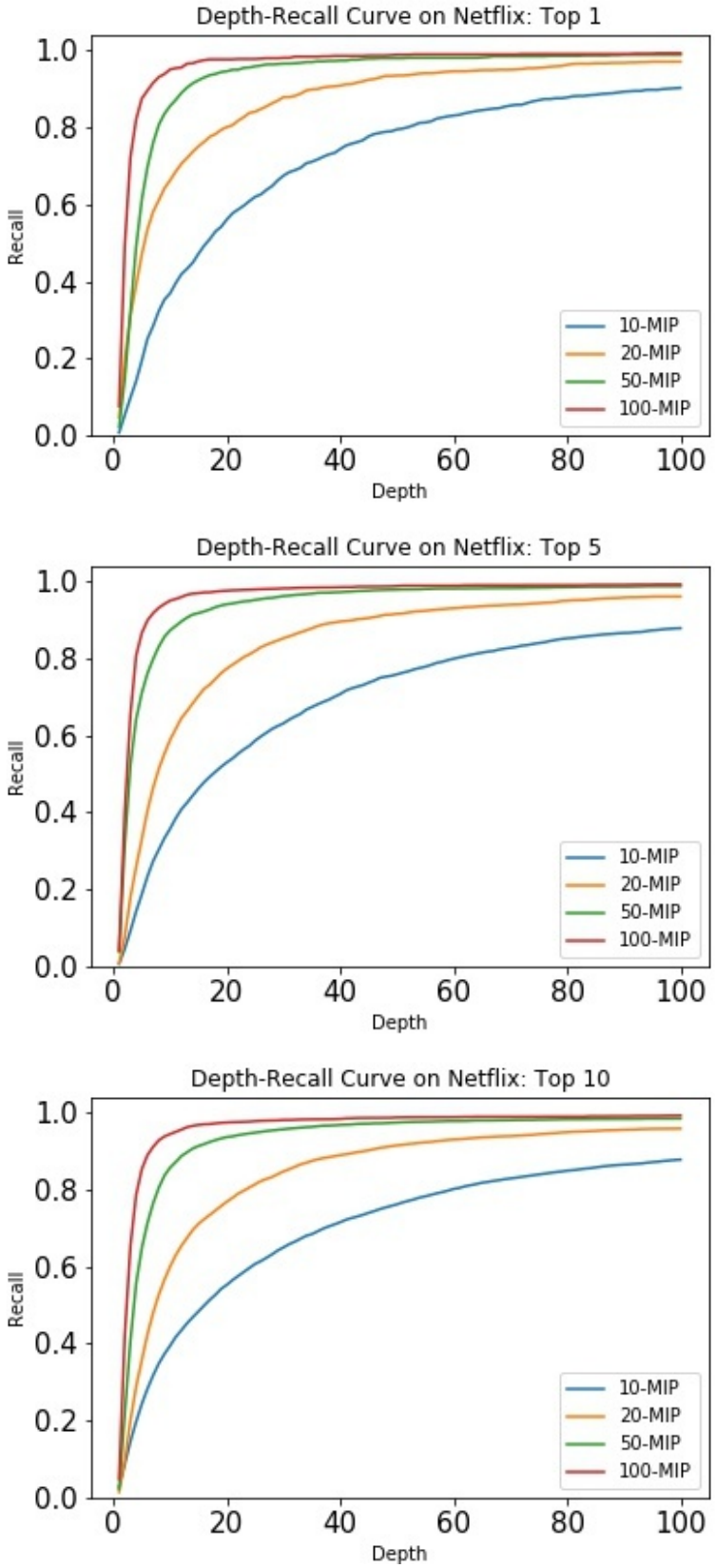
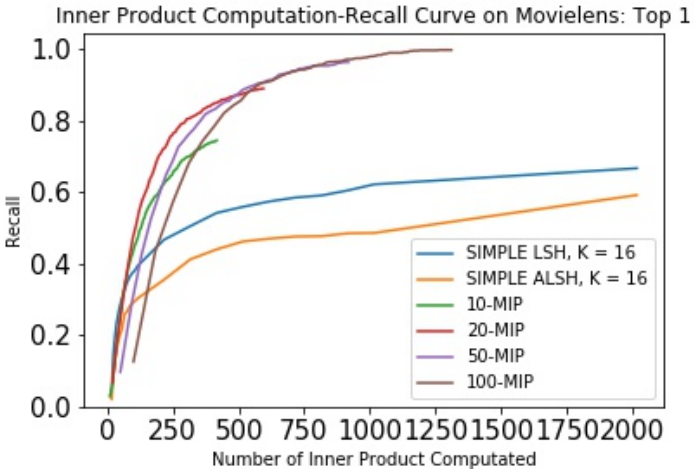
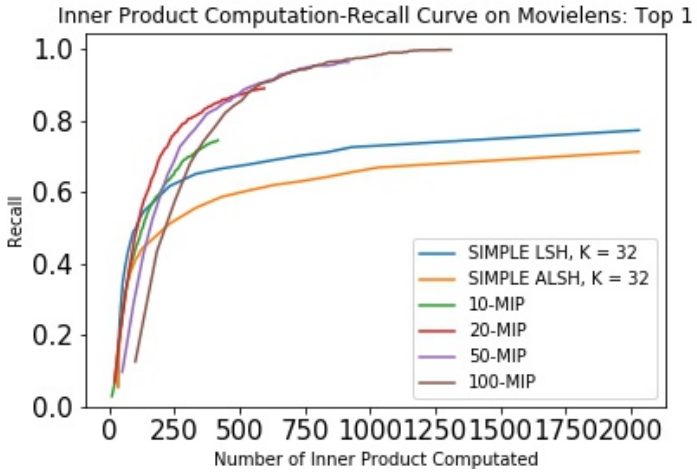
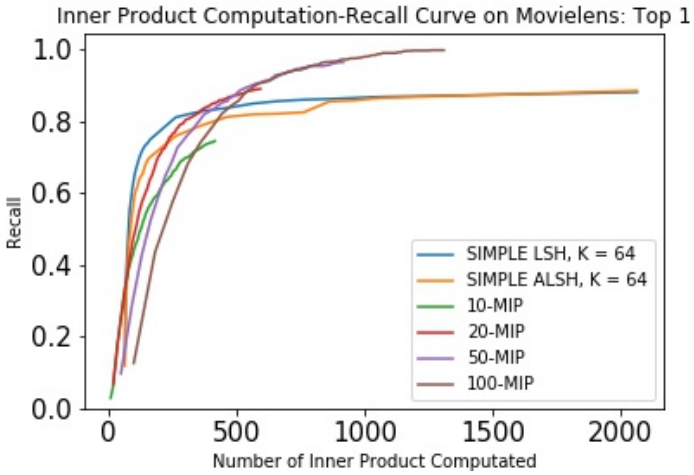
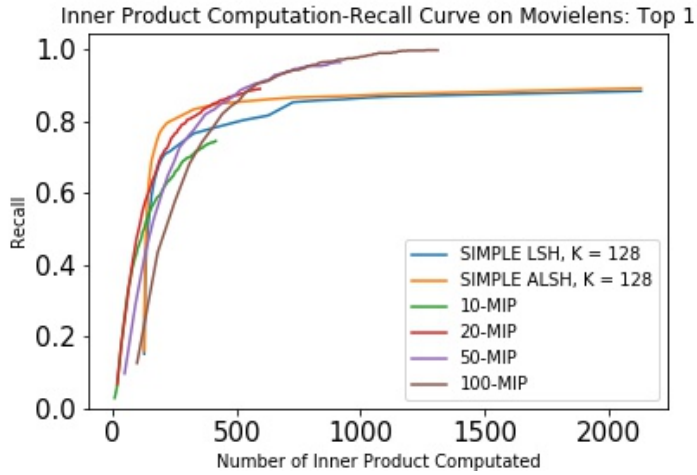


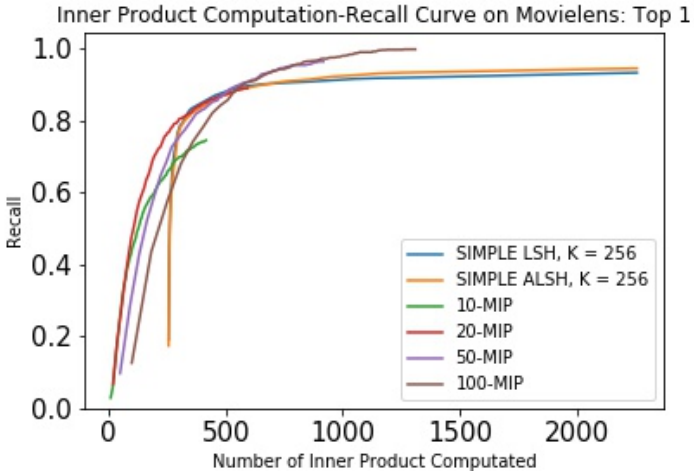
Figure 7. Computation-Recall curve of K-MIP on Movielens 10M dataset.

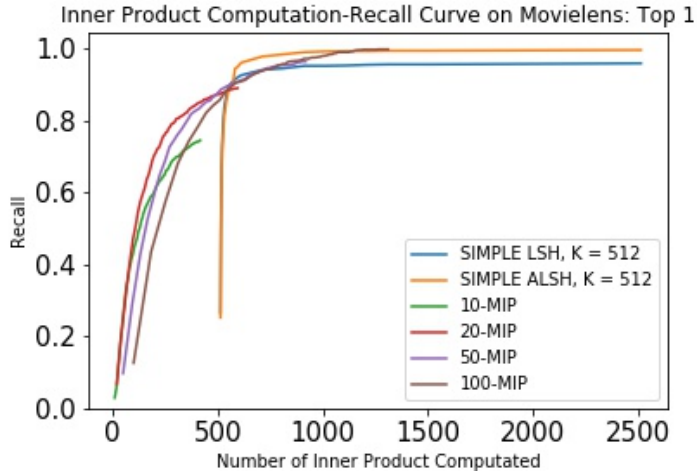


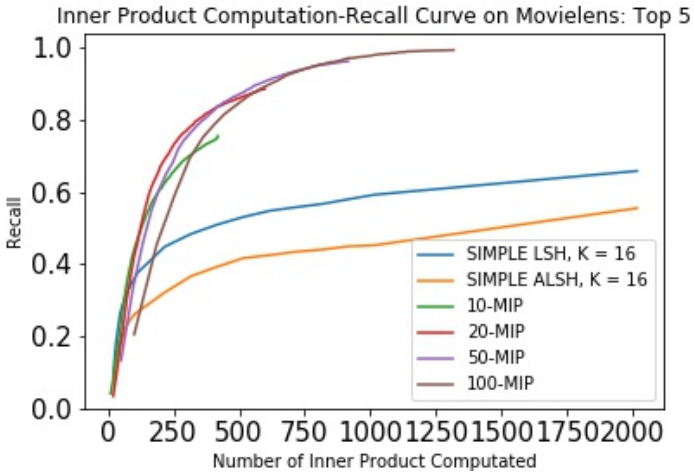


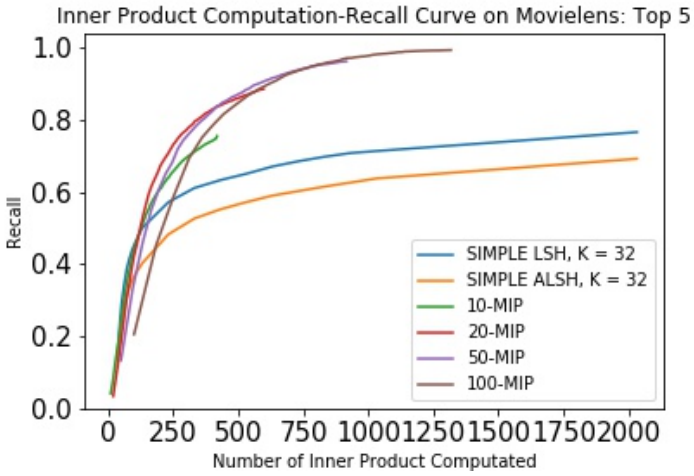


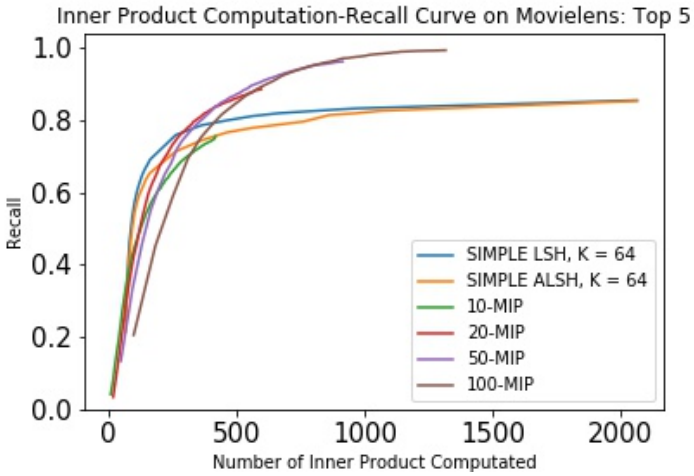


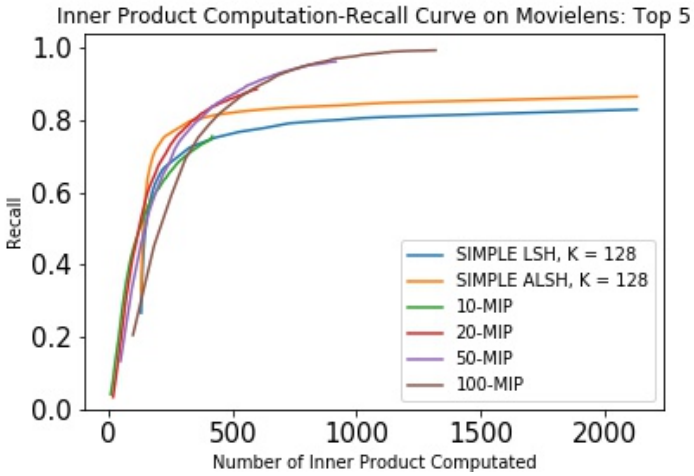


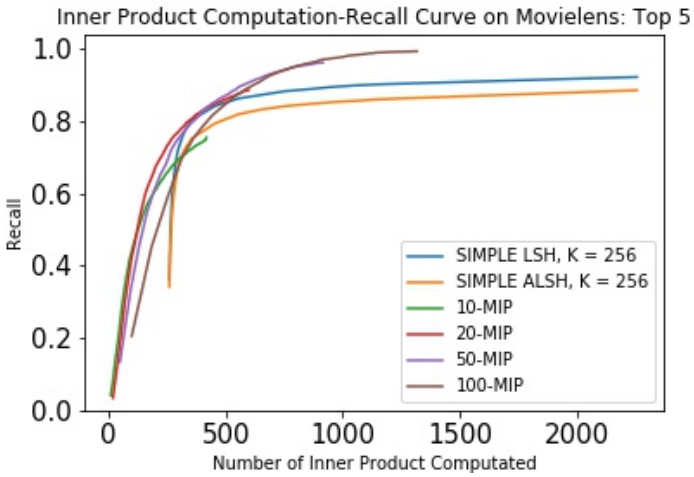


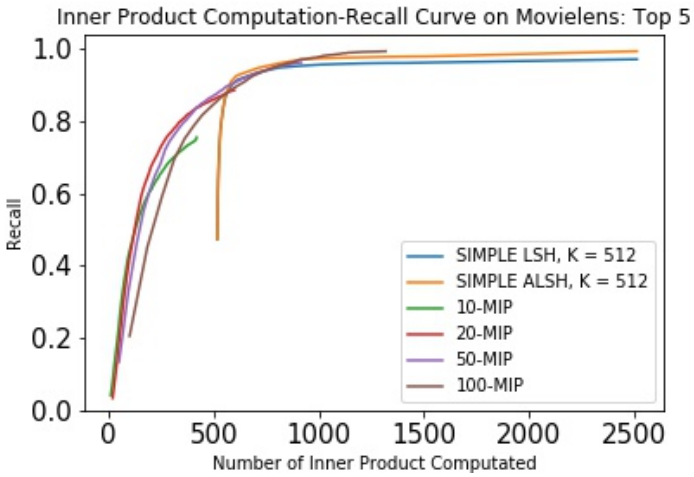

Inner Product Computation-Recall Curve on Movielens: Top 5
Recall
Number of Inner Product Computated
SIMPLE LSH, K = 16
SIMPLE ALSH, K = 16
10-MIP
20-MIP
50-MIP
100-MIP

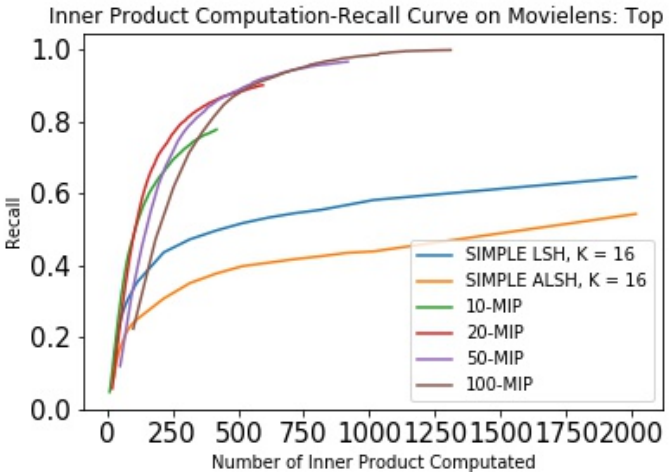

Inner Product Computation-Recall Curve on Movielens: Top 5
Recall
Number of Inner Product Computated
SIMPLE LSH, K = 32
SIMPLE ALSH, K = 32
10-MIP
20-MIP
50-MIP
100-MIP

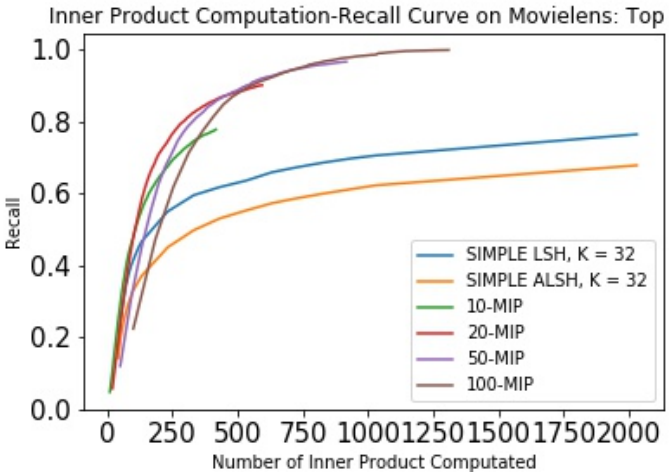

Inner Product Computation-Recall Curve on Movielens: Top 5
Recall
Number of Inner Product Computated
SIMPLE LSH, K = 64
SIMPLE ALSH, K = 64
10-MIP
20-MIP
50-MIP
100-MIP

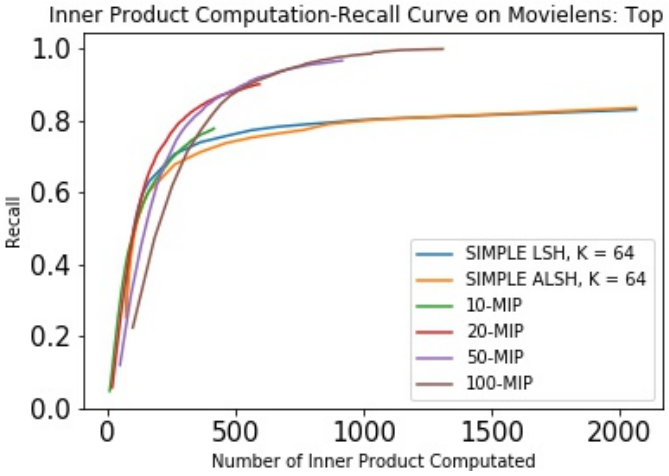

Inner Product Computation-Recall Curve on Movielens: Top 5
Recall
Number of Inner Product Computated
SIMPLE LSH, K = 128
SIMPLE ALSH, K = 128
10-MIP
20-MIP
50-MIP
100-MIP

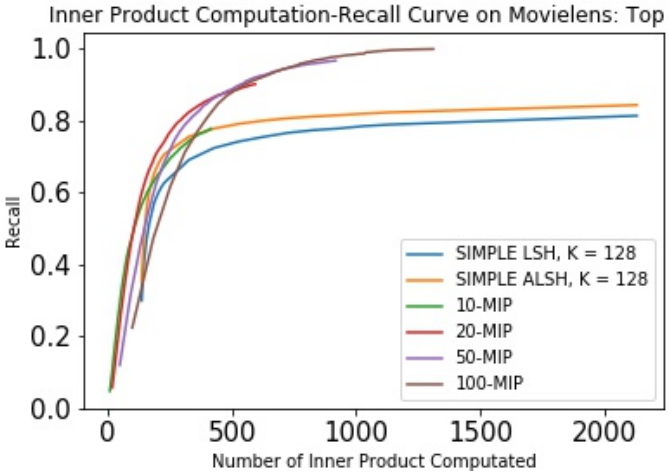

Inner Product Computation-Recall Curve on Movielens: Top 5
Recall
Number of Inner Product Computated
SIMPLE LSH, K = 256
SIMPLE ALSH, K = 256
10-MIP
20-MIP
50-MIP
100-MIP

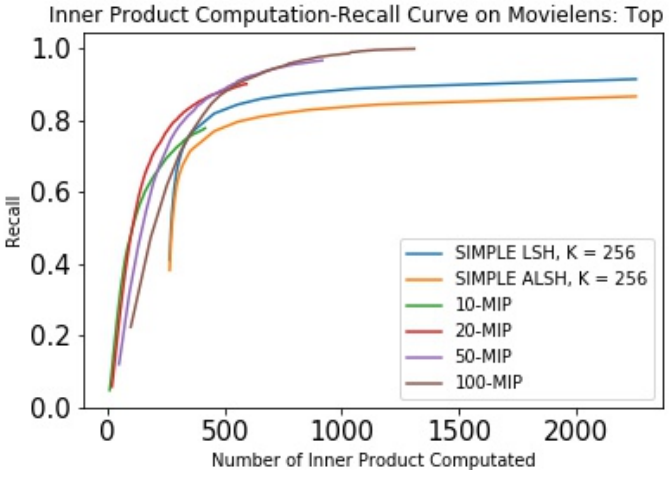

Inner Product Computation-Recall Curve on Movielens: Top 5
Recall
Number of Inner Product Computated
SIMPLE LSH, K = 512
SIMPLE ALSH, K = 512
10-MIP
20-MIP
50-MIP
100-MIP


Inner Product Computation-Recall Curve on Movielens: Top 10
Recall
Number of Inner Product Computated
SIMPLE LSH, K = 16
SIMPLE ALSH, K = 16
10-MIP
20-MIP
50-MIP
100-MIP


Inner Product Computation-Recall Curve on Movielens: Top 10
Recall
Number of Inner Product Computated
SIMPLE LSH, K = 32
SIMPLE ALSH, K = 32
10-MIP
20-MIP
50-MIP
100-MIP


Inner Product Computation-Recall Curve on Movielens: Top 10
Recall
Number of Inner Product Computated
SIMPLE LSH, K = 64
SIMPLE ALSH, K = 64
10-MIP
20-MIP
50-MIP
100-MIP


Inner Product Computation-Recall Curve on Movielens: Top 10
Recall
Number of Inner Product Computated
SIMPLE LSH, K = 128
SIMPLE ALSH, K = 128
10-MIP
20-MIP
50-MIP
100-MIP


Inner Product Computation-Recall Curve on Movielens: Top 10
Recall
Number of Inner Product Computated
SIMPLE LSH, K = 256
SIMPLE ALSH, K = 256
10-MIP
20-MIP
50-MIP
100-MIP

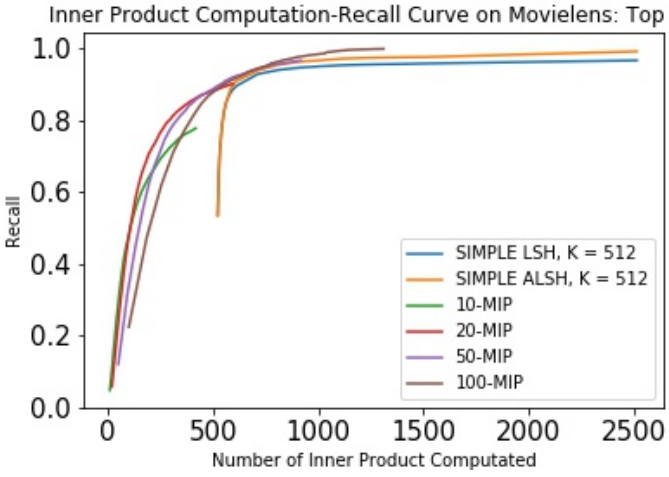
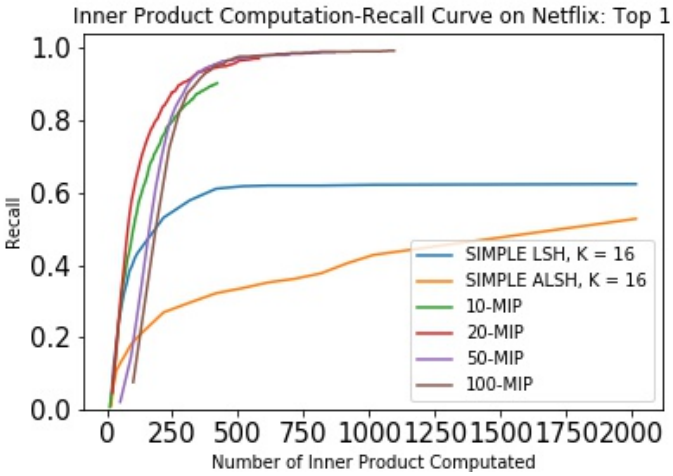
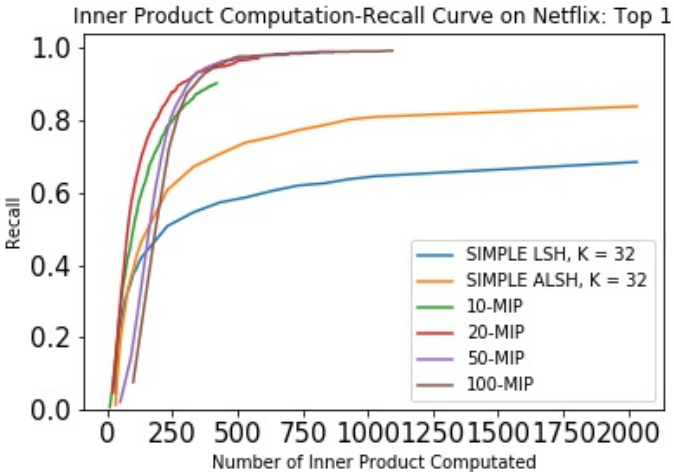
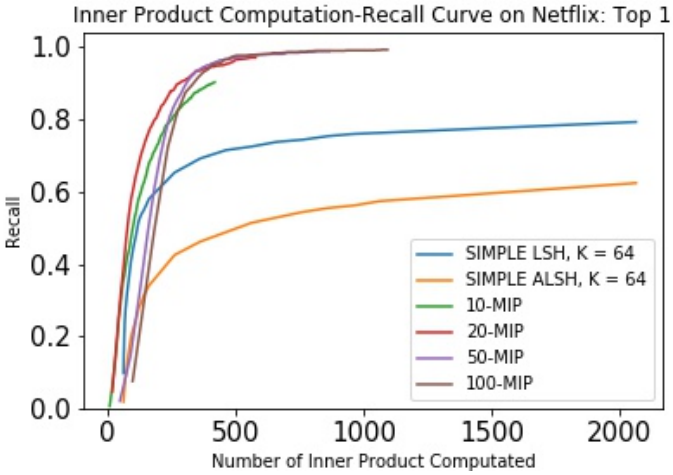
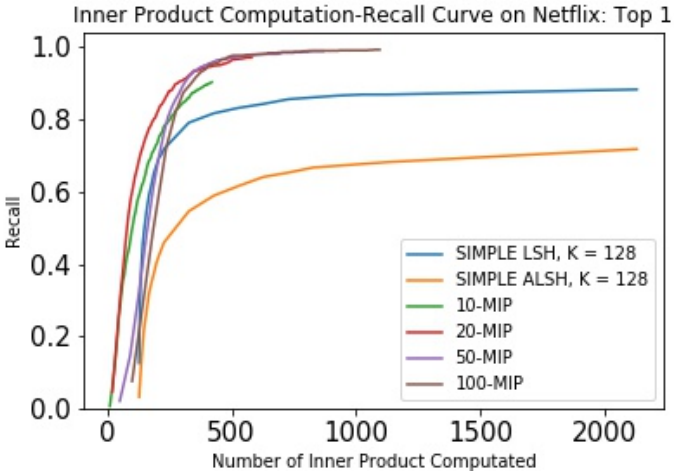

Inner Product Computation-Recall Curve on Movielens: Top 10
Recall
Number of Inner Product Computated
SIMPLE LSH, K = 512
SIMPLE ALSH, K = 512
10-MIP
20-MIP
50-MIP
100-MIP

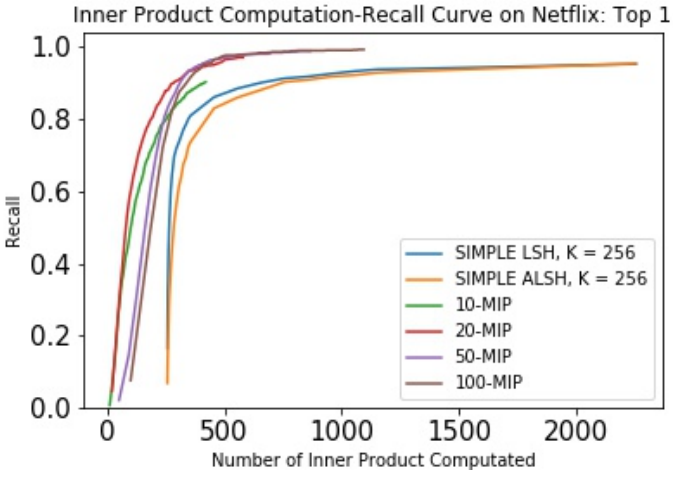
Figure 8. Computation-Recall curve of K-MIP on Netflix dataset.

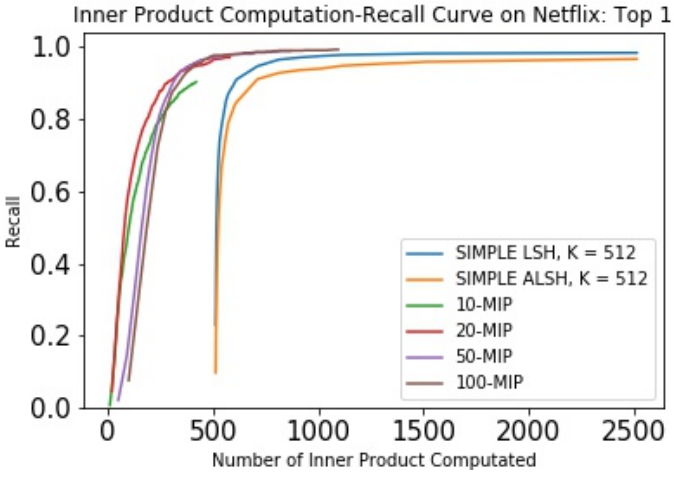


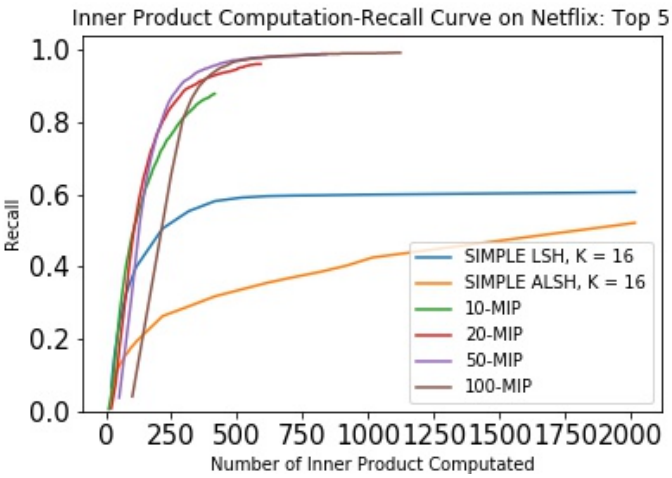


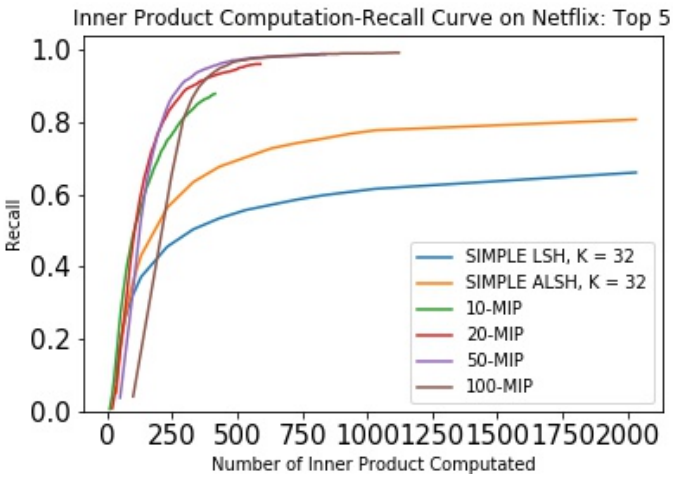


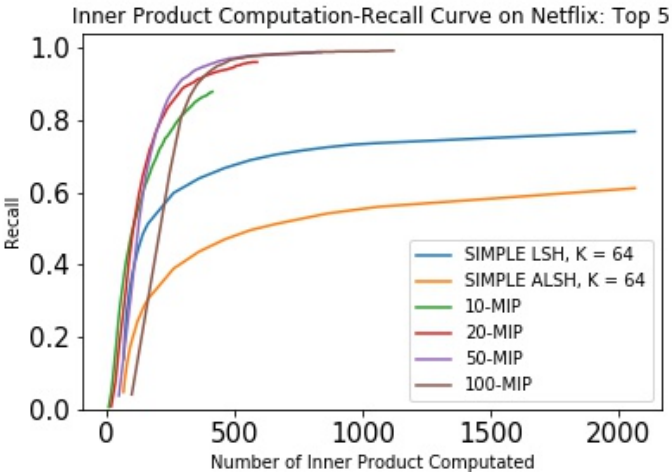


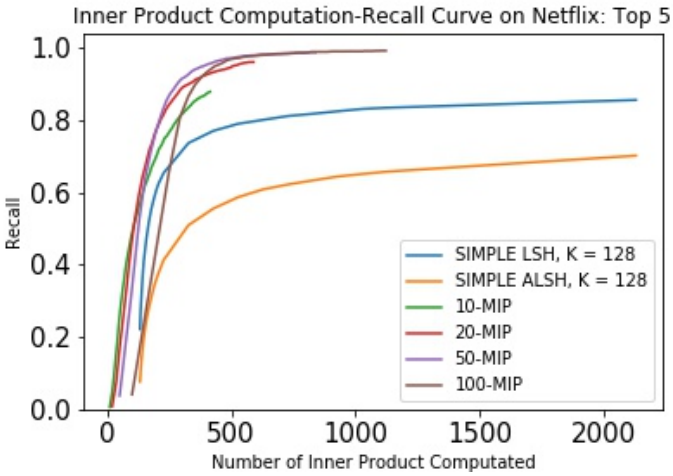


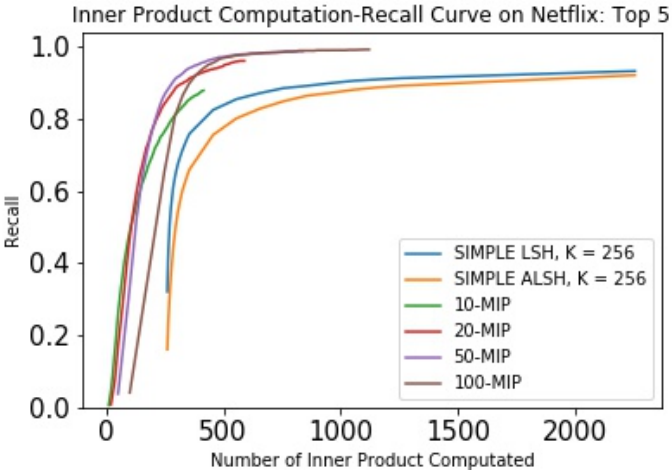


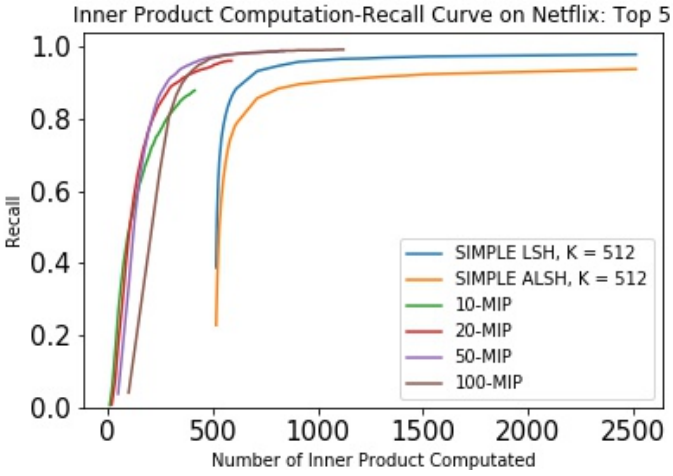

Inner Product Computation-Recall Curve on Netflix: Top 5
Recall
Number of Inner Product Computated
SIMPLE LSH, K = 16
SIMPLE ALSH, K = 16
10-MIP
20-MIP
50-MIP
100-MIP

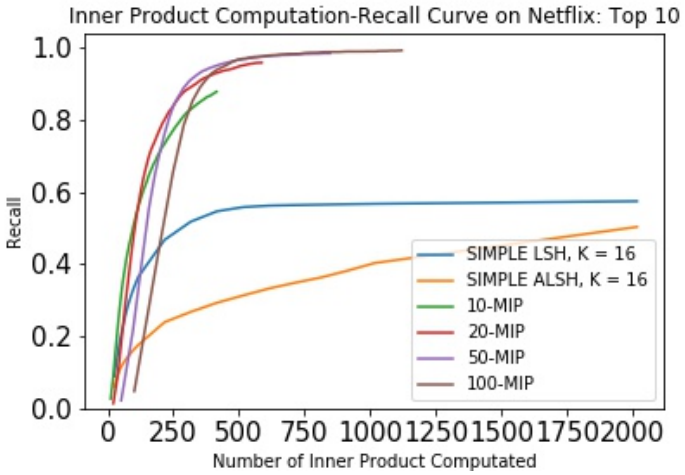

Inner Product Computation-Recall Curve on Netflix: Top 5
Recall
Number of Inner Product Computated
SIMPLE LSH, K = 32
SIMPLE ALSH, K = 32
10-MIP
20-MIP
50-MIP
100-MIP

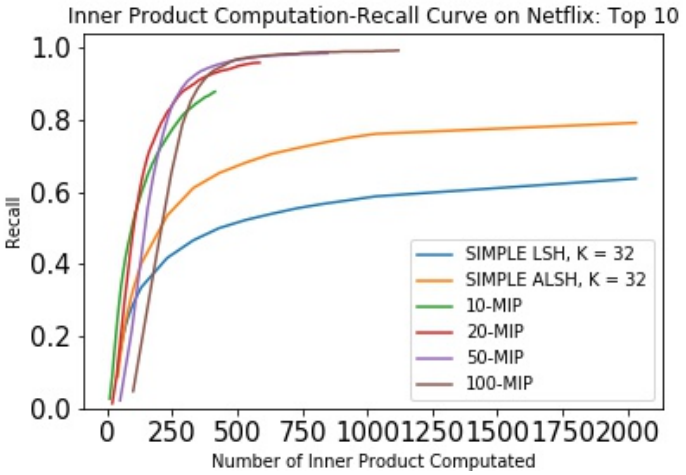

Inner Product Computation-Recall Curve on Netflix: Top 5
Recall
Number of Inner Product Computated
SIMPLE LSH, K = 64
SIMPLE ALSH, K = 64
10-MIP
20-MIP
50-MIP
100-MIP

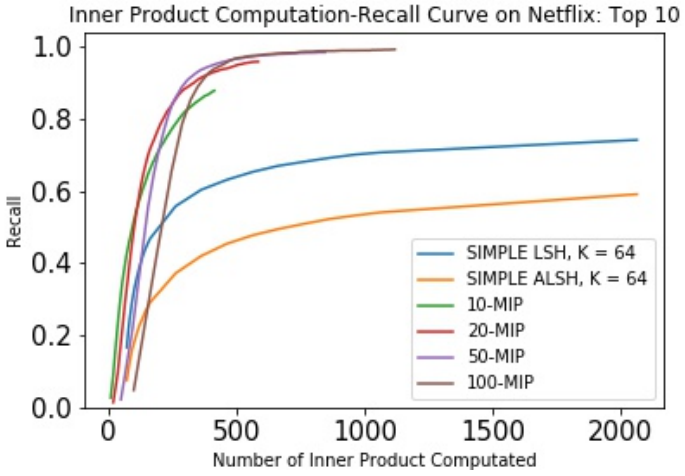

Inner Product Computation-Recall Curve on Netflix: Top 5
Recall
Number of Inner Product Computated
SIMPLE LSH, K = 128
SIMPLE ALSH, K = 128
10-MIP
20-MIP
50-MIP
100-MIP

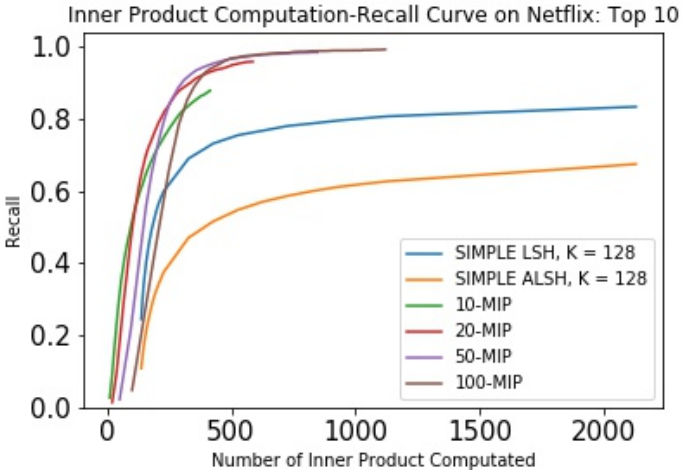

Inner Product Computation-Recall Curve on Netflix: Top 5
Recall
Number of Inner Product Computated
SIMPLE LSH, K = 256
SIMPLE ALSH, K = 256
10-MIP
20-MIP
50-MIP
100-MIP

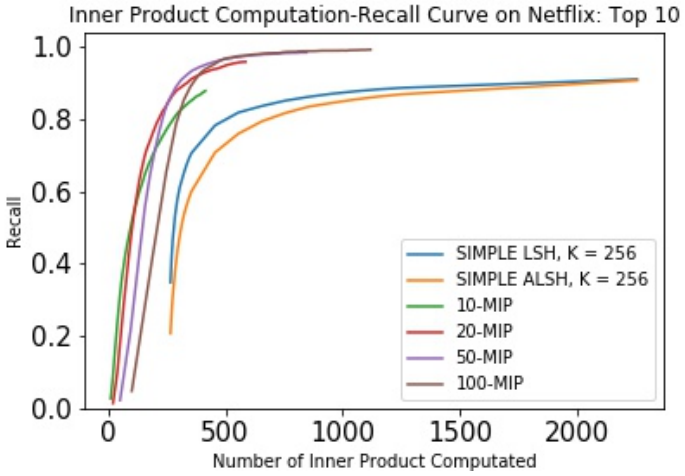

Inner Product Computation-Recall Curve on Netflix: Top 5
Recall
Number of Inner Product Computated
SIMPLE LSH, K = 512
SIMPLE ALSH, K = 512
10-MIP
20-MIP
50-MIP
100-MIP

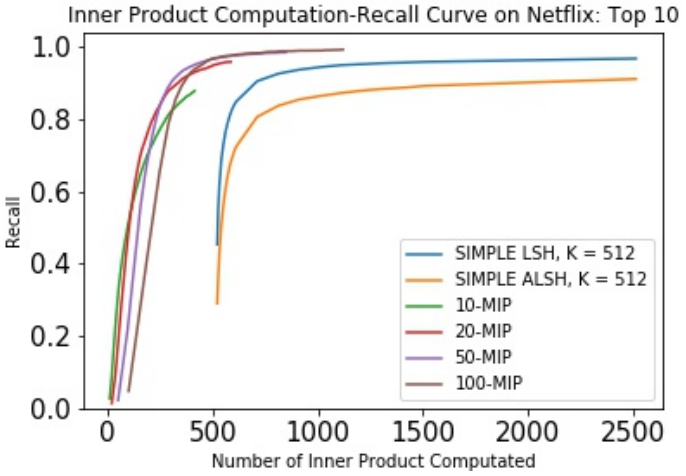

Inner Product Computation-Recall Curve on Netflix: Top 10
Recall
Number of Inner Product Computated
SIMPLE LSH, K = 16
SIMPLE ALSH, K = 16
10-MIP
20-MIP
50-MIP
100-MIP


Inner Product Computation-Recall Curve on Netflix: Top 10
Recall
Number of Inner Product Computated
SIMPLE LSH, K = 32
SIMPLE ALSH, K = 32
10-MIP
20-MIP
50-MIP
100-MIP


Inner Product Computation-Recall Curve on Netflix: Top 10
Recall
Number of Inner Product Computated
SIMPLE LSH, K = 64
SIMPLE ALSH, K = 64
10-MIP
20-MIP
50-MIP
100-MIP


Inner Product Computation-Recall Curve on Netflix: Top 10
Recall
Number of Inner Product Computated
SIMPLE LSH, K = 128
SIMPLE ALSH, K = 128
10-MIP
20-MIP
50-MIP
100-MIP


Inner Product Computation-Recall Curve on Netflix: Top 10
Recall
Number of Inner Product Computated
SIMPLE LSH, K = 256
SIMPLE ALSH, K = 256
10-MIP
20-MIP
50-MIP
100-MIP


Inner Product Computation-Recall Curve on Netflix: Top 10
Recall
Number of Inner Product Computated
SIMPLE LSH, K = 512
SIMPLE ALSH, K = 512
10-MIP
20-MIP
50-MIP
100-MIP

REFERENCES

1. Wei Dong, Charikar Moses, and Kai Li. Efficient K-Nearest Neighbor Graph Construction for Generic Similarity Measures. *WWW 11*, pages 577–586, 2011.

2. M.Datar, N.Immorlica, P.Indyk, and V.S.Mirrokn. Locality-sensitive hashing scheme based on p-stable distributions. *In SCG*, pages 253–262, Brroklyn, NY, 2004.

3.C.Fu, C.Wang and D.Cai. Fast Approximate Nearest Neighbor Search With The Vaigating Spreading-out Graph. 2018.

4.T.Ge, K.He, Q.Ke, and J.Sun. Optimized Product Quantization. *CVPR*, 2013.

5. Y.Gong and S.Lazebnik. Iterative Quantization: A Procrustean Approach to Learning Binary Codes. *CVPR*, 2011.

6. A.Malkov and A.Yashunin. Efiicient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs

7. B.Neyshabur and N.Srebro. On Symmetric and Asymmetric LSHs for Inner Product Search. *ICML*, 2015

8. A.Shrivastava and P.Li. Asymmetric LSH for Sublinear Time Maximum Inner Product Search(MIPS). *stat*.*ML*, 2014.

9.Jinfeng Li, Xiao Yan, Jian Zhang, An Xu, James Cheng, Jie Liu, Kelvin K. W. Ng, Ti-chung Cheng: A General and Efficient Querying Method for Learning to Hash. *SIGMOD Conference* 2018: 1333-1347