LSH Based Outlier Detection and Its Application in Distributed Setting*

Madhuchand Rushi Pillutla, Nisarg Raval, Piyush Bansal,
Kannan Srinathan and C. V. Jawahar
International Institute of Information Technology Hyderabad, India
{rushi.pillutla, nisarg.raval, piyush_bansal}@research.iiit.ac.in
{srinathan, jawahar}@iiit.ac.in

ABSTRACT

In this paper, we give an approximate algorithm for distance based outlier detection using Locality Sensitive Hashing (LSH) technique. We propose an algorithm for the centralized case wherein the entire dataset is locally available for processing. However, in case of very large datasets collected from various input sources, often the data is distributed across the network. Accordingly, we show that our algorithm can be effectively extended to a constant round protocol with low communication costs, in a distributed setting with horizontal partitioning.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications

General Terms

Algorithms, Performance

1. INTRODUCTION

Our outlier detection scheme is based on the definition introduced by Knorr *et al.* [2], which says that an object o is an outlier if a very large fraction p_t of the total objects in the dataset D lie outside the radius d_t from o. In our approach, we use the converse of this idea and say an object to be a non-outlier if it has enough neighbors within the radius d_t. This way, all those non-outliers in the data which have many near neighbors can be identified easily, without calculating the distances to every other object in the dataset. Moreover, in our approach whenever we identify a non-outlier we will be able to say most of its neighbors as non-outliers without even considering those objects separately. Thus, we obtain a very efficient pruning technique where, most of the non-outliers in the dataset can be easily pruned and a very small percent of the objects in the dataset need to be processed after pruning. In order to efficiently find the near neighbors in a large dimensional data, we use the Locality Sensitive Hashing(LSH) scheme. The idea of Locality Sensitive Hashing was first introduced in [1]. The basic concept of LSH is to hash all the objects such that similar objects are hashed to the same bin with high probability. Mathematically, this idea is formalized as follows [1]:

Definition 1. A family $H = h : S \rightarrow U$ is called (r_1, r_2, p_1, p_2) - sensitive if for any two objects p, q in S:

$$if\ d(p,q) \leq r_1 : Pr[h(p) = h(q)] \geq p_1 \qquad (1)$$

$$if\ d(p,q) \geq r_2 : Pr[h(p) = h(q)] \leq p_2 \qquad (2)$$

where $d(p,q)$ is the distance between objects p and q. For the hashing scheme to be locality sensitive, two conditions to be satisfied are $r_2 > r_1$ and $p_2 < p_1$. In order to amplify the gap between the probabilities p_1 and p_2, standard practice is to concatenate several hash functions to obtain a function family $G = \{g : S \rightarrow U^k\}$ such that $g(p) = (h_1(p), h_2(p), ..., h_k(p))$; where k is the width of each hash function and $h_i \in H$. For a hash function family G, the probabilities in Equation 1 and 2 are modified as:

$$if\ d(p,q) \leq r_1 : Pr[g(p) = g(q)] \geq p_1^k \qquad (3)$$

$$if\ d(p,q) \geq r_2 : Pr[g(p) = g(q)] \leq p_2^k \qquad (4)$$

During LSH, each object $o \in D$ is stored in the bins $g_j(o)$ for $j = 1, 2...L$; where each g is drawn independently and uniformly at random from G i.e., each object is hashed using L hash functions drawn from G and stored in the corresponding bins. Recently Wang *et al.* [4] proposed an outlier detection method using LSH, but unlike our approach their scheme uses the definition of an outlier given by [3].

2. ALGORITHMS

In this section we describe our algorithms for both centralized as well as distributed scheme in detail.

2.1 Centralized Setting

The algorithm for outlier detection in the centralized setting takes as input the dataset D, distance threshold d_t, point threshold p_t and outputs the outlier set M. The algorithm for centralized setting is executed in three phases.

In the First phase, initially the parameters $R = r_1 = d_t/(1+\epsilon)$ and $p'_t = (1-p_t) \times |D|$ are computed. Here $\epsilon > 0$ is an approximation factor. The LSH scheme is applied on the dataset with the parameter R, the output of which is an $L \times H$ array of bins with each bin containing the set of objects hashed to that bin. In the Second phase *ApproximateOD*

*This work was partly supported by Naval Research Board, India.

Algorithm 1 CentralizedOD

```
Input: Dataset $D$, Distance Threshold $d_t$, Point Threshold $p_t$, Threshold $\epsilon$
Output: Outliers $M$
1: $p'_t = (1 - p_t) \times |D|$
2: $R = r_1 = d_t/(1 + \epsilon)$
3: $T_{L\times H} = LSH(D, R)$
4: Compute $b_t$ // using Equation 5
5: $M' = ApproximateOD(D, T_{L\times H}, p'_t, b_t)$
6: $M = BrutForceOD(D, M', d_t, p_t)$
```

Algorithm 2 ApproximateOD

```
Input: Dataset $D$, Hash Table $T_{L\times H}$, Point Threshold $p'_t$, Bin Threshold $b_t$,
Output: Outliers $M'$
1: $M' = \{\}$
2: $\forall o \in D$, $flag[o] = 0$ // mark all objects as outliers
3: for each object $o$ in $D$ do
4:     if $flag[o] = 0$ then
5:         $l_o = \{g_i(o)|i = 1$ to $L\}$
6:         $S' = FindNeighbors(D, T_{L\times H}, o, l_o, b_t)$
7:         if $|S'| > p'_t$ then
8:             $\forall o' \in S'$, $flag[o'] = 1$ // mark objects in $S'$ as non-outliers
9:         else
10:            $M' = M' \cup \{o\}$ // add current object into set of probable outliers $M'$
11:        end if
12:    end if
13: end for
```

(Algorithm 2), using the LSH binning structure, most of the non-outliers in the dataset are pruned. We consider an object o and find all L bins to which it is hashed by LSH. We then construct the set S of all the objects stored in those L bins (without removing duplicates). The objects in this set are the probable neighbors of o. More precisely, from Equation 4, we know that each object in S is within the distance $r_2 = (1 + \epsilon) \times r_1 = d_t$ from o, with a probability at least $(1 - p_2^k)$. To boost this probability, we consider only those objects which are repeated at least $b_t(b_t \leq L)$ times in the L bins and store only those objects in a new set S'. In other words, we are reducing the error probability of considering an actual non-neighbor as a neighbor of the object o. Here, b_t is a *bin threshold* which can be computed based on the desired false negative probability as explained latter in this section.

Algorithm 3 FindNeighbors

```
Input: Dataset $D$, Hash Table $T_{L\times H}$, object $o$, hash labels $l_o$, Bin Threshold $b_t$
Output: Set $S'$ of objects satisfying Bin Threshold criteria
1: $S = S' = \{\}$
2: for $i = 1$ to $|l_o|$ do
3:     $S = S \| T[i, l_o[i]]$
4: end for
5: $\forall o' \in D$, $count[o'] = 0$
6: for each object $o' \neq o$ in $S$ do
7:     $count[o'] = count[o'] + 1$
8:     if $count[o'] \geq b_t$ then
9:         $S' = S' \cup \{o'\}$
10:    end if
11: end for
```

If the cardinality of S', returned by the *FindNeighbors* (Algorithm 3) protocol is greater than the modified point threshold $p'_t = (1 - p_t) \times |D|$, with a very high probability o cannot be an outlier since it has sufficient neighbors within distance d_t. Moreover, this holds true for all the objects in S' i.e., every object other than o also has more than p'_t neighbors with in the distance d_t, so it can not be an outlier (with a very high probability). Hence, all the objects in S' are marked as non outliers. If, on the other hand the cardinality of S' is less than or equal to the point threshold p'_t, we consider the object o as a probable outlier. This procedure is repeated till all the objects are either marked as non-outliers or probable outliers. We denote the number of objects for which the *FindNeighbors* protocol is queried as N_q and in the Section 3, we show that this value is very less (less than 1%) compared to the total dataset size. The resulting set of probable outliers M' can have a few false positives and negligible amount of false negatives. In the final phase *BruteForceOD* (Algorithm 4), the false positives are removed and the final set of approximate outliers M is returned.

The overall computational complexity of Algorithm 1 is $O(ndL)$; where $n = |D|$, d is the dimensionality of the dataset and $L = n^{1/(1+\epsilon)}$.

Algorithm 4 BruteForceOD

```
Input: Dataset $D$, Probable Outliers $M'$, Distance Threshold $d_t$, Point Threshold $p_t$
Output: Outliers $M$
1: $M = \{\}$
2: for each object $o'$ in $M'$ do
3:     $count = 0$
4:     for each object $o$ in $D$ do
5:         if $Dist(o', o) > d_t$ then
6:             $count = count + 1$
7:         end if
8:     end for
9:     if $count \geq p_t$ then
10:        $M = M \cup \{o'\}$
11:    end if
12: end for
```

False Positives and False Negatives: In the context of outlier detection, a false positive (fp) is to label a non-outlier as an outlier and false negative (fn) is to label an outlier to be a non-outlier. In this section we give a bound on the probabilities of both based on the value of the bin threshold parameter b_t. Consider a LSH scheme where each object is hashed using L hash functions each of width k. The probability of two objects at a distance greater than $r_2 = r_1 \times (1 + \epsilon) = d_t$, to be hashed to the same bin is at most p_2^k. Consider the worst case scenario where an actual outlier has exactly p'_t neighbors. Hence, counting one non-neighbor as a neighbor will lead to a false negative. In our scheme, for a non-neighbor to be counted as a neighbor of an object o, it should hash to the same bin as o for at least b_t times out of L times. The probability that this happens for exactly b_t times out of L is given by the binomial probability:

$$Pr'[fn] \leq \binom{L}{b_t} P_2^{kb_t}(1 - P_2^{kb_t})^{L-b_t}$$

Hence, probability for a false negative is upper bounded by:

$$Pr[fn] < (L - b_t) \times \binom{L}{b_t} P_2^{kb_t}(1 - P_2^{kb_t})^{L-b_t} \quad (5)$$

Similarly, the probability that two objects within a distance R being hashed to two different bins is at most $(1-p_1^k)$. The probability that this will happen at least $(L - b_t)$ times out of L times is given as:

$$Pr[fp] < b_t \times \binom{L}{L - b_t}(1 - p_1^{k(L-b_t)})p_1^{kb_t} \quad (6)$$

This gives an upper bound for the probability of a false positive.

Bin Threshold: As seen from the Equations 5 and 6, both the false negative and false positive probabilities depend on the bin threshold. Increasing the bin threshold has an effect of decreasing the false negatives at the cost of an increase in the false positives and vise versa. An optimal value for b_t would be one which would remove the false negatives at the cost of introducing minimal false positives. In our scheme, the user has the flexibility to fix b_t, based on the false negative probability desired, using Equations 5.

2.2 Distributed Setting

We consider the horizontal distribution where each player has a subset of the total number of objects. The distributed algorithm is given in two player setting, which can be easily extended to a p player setting. Consider two players denoted by P^A and P^B with local datasets D^A and D^B respectively.

Algorithm 5 DistributedOD

```
Input: Player $P^A$ with Dataset $D^A$, Player $P^B$ with Dataset $D^B$, Distance Threshold $d_t$, Point Threshold $p_t$, Threshold $\epsilon$
Output: Player $P^A$ Outliers $M^A$
 1: At $P^A$ :
 2: $P^A$ sends $|D^A|$ to $P^B$
 3: At $P^B$ :
 4: $n = |D^A| + |D^B|$
 5: $P^B$ sends $n$ to $P^A$
 6: At $P^A$ :
 7: $p'_t = (1 - p_t) \times n$
 8: $R = d_t/(1 + \epsilon)$
 9: $T^A_{L\times H} = LSH(D^A, R)$
10: compute $b_t$
11: $M'^A = ApproximateOD(D^A, T^A_{L\times H}, p'_t, b_t)$
12: $M''^A = GlobalApproxOD(M'^A)$
13: $M^A = GloabalOD(M''^A)$
```

We present the algorithm such that one player, say P^A will be able to compute its subset of the global outliers at the end of the algorithm. Similarly the algorithm can be used to enable P^B to compute its subset of the global outliers by simply interchanging the roles of P^A and P^B in the algorithm. Before giving a detailed description of the algorithm, we give the following definitions.

Definition 2. local outlier: Given a distance threshold d_t and a point threshold p_t, an object o with player P^i is a local outlier if the number of objects in the local dataset D^i lying at a distance greater than d_t is at least a fraction p_t of the total dataset D.

Definition 3. global outlier: Given a distance threshold d_t and a point threshold p_t, an object o in a dataset D is a global outlier if at least fraction p_t of the objects in D lie at a distance greater than d_t from o.

Algorithm 6 GlobalApproxOD

```
Input: Players $P^A$ and $P^B$, Player $P^A$ Dataset $D^A$, Player $P^B$ Dataset $D^B$, Point Threshold $p'_t$, Bin Threshold $b_t$, Player $P^A$ Hash Table $T^A_{L\times H}$, Player $P^B$ Hash Table $T^B_{L\times H}$, Player $P^A$ Local Approximate Outliers $M'^A$
Output: Player $P^A$ Global Approximate Outliers $M''^A$
 1: At $P^A$ :
 2: for each object $o$ in $M'^A$ do
 3:     $l^A_o = \{g_i(o)|i = 1 \text{ to } L\}$
 4: end for
 5: send $l^A$ to $P^B$
 6: At $P^B$ :
 7: $\forall l \in l^A$, $c^B[l] = 0$
 8: for each label $l$ in $l^A$ do
 9:     $S' = FindNeighbors(D^B, T^B_{L\times H}, null, l, b_t)$
10:     $c^B[l] = |S'|$
11: end for
12: send $c^B$ to $P^A$
13: At $P^A$ :
14: $M''^A = \{\}$
15: for each object $o'$ in $M'^A$ do
16:     $l_o = \{g_i(o)|i = 1 \text{ to } L\}$
17:     $S' = FindNeighbors(D^A, T^A_{L\times H}, o', l_o, b_t)$
18:     $count = |S'| + c^B[o]$
19:     if $count \le p'_t$ then
20:         $M''^A = M''^A \cup \{o'\}$
21:     end if
22: end for
```

In the first phase, player P^A sends the size of his dataset to P^B and gets the size of the entire dataset (i.e. $|D^A|$+$|D^B|$). Player P^A locally computes its local probable outliers M'^A by running the centralized algorithm on its dataset D^A. In the second phase, for each object o in the set M'^A, P^A forms the set l^A_o which is the set of L labels of the bins to which o is hashed to (these labels are stored while performing LSH). P^A sends the set l^A to P^B. Player P^B considers each element l in l^A and runs the *FindNeighbors* algorithm on its local dataset. P^B then counts the cardinality of the sets returned by *FindNeighbors* and sends the set of counts to P^A. Player P^A considers each object o in M'^A and runs the *FindNeighbors* algorithm to obtain its count of the number of objects repeating more than b_t times (this step is actually redundant if P^A stores this count in first phase). P^A then computes the sum of this count and the corresponding count in c^B, and if this sum is less than or equal to point threshold P'_t, P^A stores o in the set M''^A of global probable outliers.

The set M''^A contains some false positives which can be removed in third phase with another round of communication as follows: P^A sends the set M''^A to P^B. Player P^B considers each object o in M''^A and computes the distance to each object in its dataset D^B and counts the number of objects which lie at distance greater than the distance threshold d_t from o. P^B sends all the counts back to P^A. P^A then considers each object o in M''^A and computes the distance to each object in its dataset D^A and counts the number of objects lying at distance greater than d_t. P^A

then computes the sum of this count and the corresponding count received from P^B and if this count is greater than or equal to fraction p_t of n, marks the object o as an outlier.

The computational complexity of player P^A is $O(n^A dL)$. The overall communication complexity of player P^A would be $O(m'^A L + m''^A d)$, where $m'^A = |M'^A|$ and $m''^A = |M''^A|$.

Algorithm 7 GlobalOD

```
Input: Players $P^A$ and $P^B$, Player $P^A$ Dataset $D^A$, Player
    $P^B$ Dataset $D^B$, Distance Threshold $d_t$, Point Thresh-
    old $p_t$, Player $P^A$ Global Approximate Outliers $M''^A$
Output: Player $P^A$ Global Outliers $M^A$
 1: send $M''^A$ to $P^B$
 2: At $P^B$ :
 3: $\forall o'' \in M''^A$, $c^B[o''] = 0$
 4: for each object $o''$ in $M''^A$ do
 5:    for each object $o$ in $D^B$ do
 6:       if $Dist(o'', o) > d_t$ then
 7:          $c^B[o''] = c^B[o''] + 1$
 8:       end if
 9:    end for
10: end for
11: send $c^B$ to $P^A$
12: At $P^A$ :
13: for each object $o''$ in $M''^A$ do
14:    $count = 0$
15:    for each object $o$ in $D^A$ do
16:       if $Dist(o'', o) > d_t$ then
17:          $count = count + 1$
18:       end if
19:    end for
20:    $count = count + c^B[o'']$
21:    if $count \geq p_t$ then
22:       $M^A = M^A \cup \{o''\}$
23:    end if
24: end for
```

3. EXPERIMENTS

All experiments are performed on Intel(R) Core $i7$ CPU $3.33GHz$ machine, using various datasets[1] listed in Table 1.

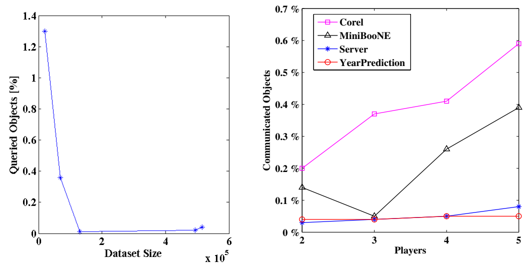
Centralized: Table 1 outline the performance of proposed *CentralizedOD* algorithm. First 3 columns lists the various datasets with respective number of objects and dimensions. The execution time (in seconds) averaged over multiple runs of our algorithm in centralized setting is tabulated in column 4. We computed the optimal bin threshold b_t (listed in column 5) as described in Section 2 for mentioned datasets and ran our algorithm using the same bin threshold. By setting bin threshold to the optimal value, we achieved very low false positive rate as shown in the column 6. The final phase of removing false positives can be avoided if this false positive rate is acceptable. Even if the false positives needs to be removed, the low false positive rate ensures that the brute force needs to process only few objects. The false negative rate is not mentioned in the table as it is 0 in case of optimal Bin Threshold. In other words, we achieved 100% detection rate on optimal bin threshold.

For each dataset considered, we computed the percentage of objects for which the *FindNeighbors* protocol is invoked.

[1]All datasets are taken from UCI Machine Learning Repository. http://archive.ics.uci.edu/ml

Table 1: Performance of CentralizedOD

Dataset	Objects	Dim	Time	b_t	FP
Letter	20000	16	7.53	2	0.12
Corel	68040	32	32.27	20	0.09
MiniBooNE	130064	50	29.86	4	0.01
Server	494021	5	41.93	22	0.02
YearPrediction	515345	90	193.11	4	0.03



(a) (b)

Figure 1: (a) Queried Objects N_q (b) Communication Cost in Horizontal Distribution

The results are shown in Figure 1(a). For small datasets the percentage of queried objects is close to 1%, but as the dataset gets larger this value keeps decreasing rapidly which is evident from the figure.

Distributed: For the distributed case, we have uniformly distributed all the objects in a dataset among the players. For all datasets, the union of the outliers found at each player is equivalent to the set of outliers returned by the centralized algorithm run on the same dataset. This shows that functionality is preserved in the distributed setting also. We have repeated the experiment by varying the number of players from 1 to 5 and studied the effect on the communication cost. The results are given in Figure 1(b). It can be seen that the percentage of the communicated objects is indeed very less.

4. CONCLUSIONS

In this paper, we have proposed an approximate algorithm for distance based outlier detection. Our approach uses the Locality sensitive hashing (LSH) to achieve an efficient pruning technique. Further, we have extended our scheme to horizontally distributed setting and proved it to be highly efficient in terms of the communication cost.

5. REFERENCES

[1] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *STOC*, pages 604–613, 1998.

[2] E. M. Knorr and R. T. Ng. Algorithms for mining distance-based outliers in large datasets. In *VLDB*, pages 392–403, 1998.

[3] S. Ramaswamy, R. Rastogi, and K. Shim. Efficient algorithms for mining outliers from large data sets. *SIGMOD Rec.*, 29:427–438, 2000.

[4] Y. Wang, S. Parthasarathy, and S. Tatikonda. Locality sensitive outlier detection: A ranking driven approach. In *Proc. ICDE 2011*, 2011.