DISSEMINATION OF SENSITIVE DATA

[1]K Bindu Susan ,[2] M.Raja Babu

[1]Department of computer Science, Aditya Engineering College,JNT University,Kakinada.

[1]Department of computer Science,JNT University, Kakinada.

Abstract

Defending sensitive data dissemination (anonymizing) from adversary's activities like Record, Attribute, Table linkages is an enforcing aspect for better prospects. Existing popular protective measures like k-anonymity and l-diversity perform better in achieving overall data utility maximization by reducing the information loss incurred in the anonymizing process. Unfortunately their strengths are confined to fixed schema data sets with low dimensionality. Earlier two novel anonymization methods such as approximate nearest-neighbor (NN) search using locality-sensitive hashing (LSH) and data transformation techniques like reduced band matrix , gray encoded sorting are used to parse high-dimensional spaces. The random projection feature of LSH is a computation overhead. So we propose to replace the former method with a variant of a k-d Tree (Spill tree) that uses an overlapping splitting area to find nearest-neighbor(NN). NN-search using Spill trees has significant performance boost with superior data utility and best execution time. We show experimentally by using both real data sets (from UCI and KDD repositories) and also synthetic data sets designed to exercise the algorithms in various ways.

1. INTRODUCTION

Privacy preserving has received considerable attention from the database community in the past few years. Existing privacy-preserving techniques focus on anonymizing personal data, which have a fixed schema with a small number of dimensions. Through *generalization* or *suppression*, existing methods prevent attackers from re-identifying individual records. Generalization is a popular method of thwarting linking attacks. It works by replacing quasi identifier(QI)-values in the microdata with fuzzier forms. let T be a table containing sensitive information. The objective is to release a modified version of T such that modified versions forbids adversaries from inferring the sensitive data of T confidently then table T is often called microdata.

A microdata relation can be generalized in numerous ways. generalization needs to be guided by an anonymization principle, which is a criterion deciding whether a table has been adequately anonymized. Most notable principles include k-anonymity , l-diversity. Existing principles work for a single sensitive attribute, whereas we need to consider a larger number of sensitive items.

Later nearest neighbor (NN) search and data transformation techniques are applied for anonymizing sensitive information. NN search is based on locality sensitive hashing which

outperforms in terms of data utility, but incurs slightly higher computational overhead.

In this paper, we replace the existing method with K-D tree variant is implemented which performs nearest neighbor search. Significant efficiency improvement has been observed comparing to LSH (localify sensitive hashing), the state of art approximate k-NN algorithm.

2. RELATED WORK

k-anonymity prevents re-identification of individual records, but it is vulnerable to *homogeneity* attacks, where many of the records in an anonymized group share the same sensitive attribute (SA) value. l-diversity addresses this vulnerability, and creates anonymized groups in which at least SA values are well-represented. Any *k*-anonymity technique can be adapted for l-diversity; however, this approach typically causes high information loss. A framework is proposed based on dimensionality mapping, which can be tailored for *k*-anonymity and *l*-diversity, and outperforms other generalization techniques. However, dimensionality mapping is only effective for low-dimensional QIDs, hence the method is not suitable for transactional data. Furthermore, existing l-diversity methods work for a single sensitive attribute, whereas in our problem, we need to consider a larger number of sensitive items. External knowledge is available to an adversary, in the form of logical constraints on data records. However, the solution proposed targets relational (i.e., low-dimensional) data.

Locality-sensitive hashing (LSH) which represents documents as bit-signatures, such that two similar documents are likely to have similar signatures. A sort-based sliding window algorithm on permutations of bit signatures is used to extract similar pairs. LSH provides a tradeoff between effciency and effectiveness by usercontrolled parameters, and can be straightforwardly parallelized since multiple randomizations run independently. weakness of LSH approaches in general is that they present a bewildering number of parameters that need to be set, and provide little guidance for an application developer approaching new problems and causes overheads.

3. INTRODUCTION TO KD-TREE

A kd-tree is a data structure for storing a finite set of points from a k-dimensional space. A kd-tree is a binary tree. The exemplar set E is represented by the set of nodes in the kd-tree, each node representing one exemplar. The dom-elt field represents the domain vector of the exemplar and the range-elt field represents the range vector. The dom-elt component is the index for the node. It splits the space into two subspaces according to the splitting hyperplane of the node. All the points in the left-subspace are represented by the left subtree and the points in the right-subspace by the right subtree. The splitting hyperplane is a plane which passes through dom-elt and which is perpendicular to the direction specified by the split field. Let i be the value of the split field. Then a point is to the left of dom-elt if and only if its ith component is less than the ith component of dom-elt. The complimentary definition holds for the right field. If a node has no children, then the splitting hyperplane is not required.

Given an exemplar set E, a kd-tree can be constructed by the algorithm. The pivot choosing procedure of Step 2 inspects the set and chooses a good domain vector from this set to use as the trees root. The discussion of how such a root is chosen. Whichever exemplar is chosen as root will not affect

Field Name:	Field Type	Description
dom-elt	domain-vector	A point from k_d-d space
range-elt	range-vector	A point from k_r-d space
split	integer	The splitting dimension
left	kd-tree	A *kd*-tree representing those points to the left of the splitting plane
right	kd-tree	A *kd*-tree representing those points to the right of the splitting plane

Table 1: The fields of a kd-tree node

the correctness of the kd-tree though the trees maximum depth and the shape of the hyperregions will be affected.

4. NEAREST NEIGHBOR SEARCH

A first approximation is initially found at the leaf node which contains the target point. The target point is marked X and the leaf node of the region containing the target is coloured black. As is exemplified in this case, this first approximation is not necessarily the nearest neighbour but at least we know any potential nearer neighbour must lie closer and so must lie within the circle centred on X and passing through the leaf node.

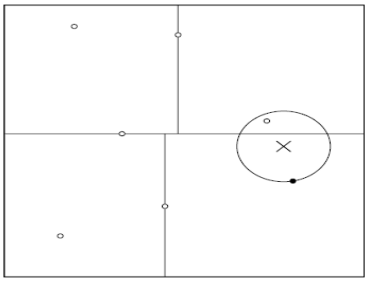

Figure 1 :The black dot is the dot which owns the leaf node containing the target (the cross). Any nearer neighbour must lie inside this circle.

We now back up to the parent of the current node. The parent is the black node, We compute whether it is possible for a closer solution to that so

Algorithm:	Nearest Neighbour in a *k*d-tree
Input:	**kd**, of type **kdtree** **target**, of type **domain vector** **hr**, of type **hyperrectangle** **max-dist-sqd**, of type **float**
Output:	**nearest**, of type **exemplar** **dist-sqd**, of type **float**
Pre:	*Is-legal-kdtree*(**kd**)
Post:	Informally, the postcondition is that **nearest** is a nearest exemplar to **target** which also lies both within the hyperrectangle **hr** and within distance $\sqrt{\textbf{max-dist-sqd}}$ of **target**. $\sqrt{\textbf{dist-sqd}}$ is the distance of this nearest point. If there is no such point then **dist-sqd** contains infinity.
Code:	
1.	if **kd** is empty then set **dist-sqd** to infinity and exit.
2.	**s** := **split** field of **kd**
3.	**pivot** := **dom-elt** field of **kd**
4.	Cut **hr** into two sub-hyperrectangles **left-hr** and **right-hr**. The cut plane is through **pivot** and perpendicular to the **s** dimension.
5.	**target-in-left** := $\textbf{target}_\textbf{s} \leq \textbf{pivot}_\textbf{s}$
6.	if **target-in-left** then
6.1	**nearer-kd** := **left** field of **kd** and **nearer-hr** := **left-hr**
6.2	**further-kd** := **right** field of **kd** and **further-hr** := **right-hr**
7.	if **not target-in-left** then
7.1	**nearer-kd** := **right** field of **kd** and **nearer-hr** := **right-hr**
7.2	**further-kd** := **left** field of **kd** and **further-hr** := **left-hr**
8.	Recursively call **Nearest Neighbour** with parameters (**nearer-kd**,**target**, **nearer-hr**,**max-dist-sqd**), storing the results in **nearest** and **dist-sqd**
9.	**max-dist-sqd** := minimum of **max-dist-sqd** and **dist-sqd**
10.	A nearer point could only lie in **further-kd** if there were some part of **further-hr** within distance $\sqrt{\textbf{max-dist-sqd}}$ of **target**. if this is the case then
10.1	if $(\textbf{pivot} - \textbf{target})^2 < \textbf{dist-sqd}$ then
10.1.1	**nearest** := (**pivot**, **range-elt** field of **kd**)
10.1.2	**dist-sqd** := $(\textbf{pivot} - \textbf{target})^2$
10.1.3	**max-dist-sqd** := **dist-sqd**
10.2	Recursively call **Nearest Neighbour** with parameters (**further-kd**,**target**, **further-hr**,**max-dist-sqd**), storing the results in **temp-nearest** and **temp-dist-sqd**
10.3	If **temp-dist-sqd** < **dist-sqd** then
10.3.1	**nearest** := **temp-nearest** and **dist-sqd** := **temp-dist-sqd**

Nearest Neighbor Algorithm in KD-tree.

far found to exist in this parents other child. Here it is not possible because the circle does not intersect with the shaded space occupied by the parents other child. If no closer neighbour can exist in the other child, the algorithm can immediately move up a further level else it must recursively explore the other child.

In this example, the next parent which is checked will need to be explored, because the area it covers i.e.,everywhere north of the central horizontal line does intersect with the best circle so far. The search will only take place within those portions of the kd –tree which lie both in the hyper rectangle, and within the maximum distance to the target. The caller of the routine will generally specify the in_nite hyperrectangle which covers the whole of Domain, and the infinite maximum distance.

The search is depth first and uses the heuristic of searching first the child node which contains the target. Step 1 deals with the trivial empty tree case, and Steps 2 and 3 assign two important local variables. Step 4 cuts the current hyperrectangle into the two hyperrectangles covering the space occupied by the child nodes. Steps 5-7 determine which child contains the target. After Step 8, when this initial child is searched, it may be possible to prove that there cannot be any closer point in the hyperrectangle of the further child. In particular, the point at the current node must be out of range. The test is made in Steps 9 and 10. Step 9 restricts the maximum radius in which any possible closer point could lie, and then the test in Step 10 checks whether there is anyspace in the hyperrectangle of the further child which lies within this radius. If it is not possible then no further search is necessary. If it is possible, then Step 10.1 checks if the point associated with the current node of the tree is closer than the closest yet. Then, in Step 10.2 the further child is recursively searched. The maximum distance worth examining in this further search is the distance to the closest point yet discovered.

The proof that this will find the nearest neighbour within the constraints is by induction on thesize of the kd-tree. If the cutoff were not made in Step 10 then the proof would be straightforward: the point returned is the closest out of (i) the closest point in the nearer child. (ii) the point at the current node and (iii) the closest point in the further child. If the cutoff were made in Step 10, then the point returned is the closest point in the nearest child, and we can show that neither the current point, nor any point in the further child can possibly be closer.

5. PERFORMANCE

The kNN algorithms we consider include an approximation parameter, which affects their accuracy. the kd-trees include an approximation factor Q, and the LSH structure includes a success probability p_s. For spill-trees, we can vary the spill buffer size τ as well as the threshold t_m that determines whether each node is a spill node or a metric node. We analyze the performance of the kNN algorithms on the given dataset as we vary these parameters.

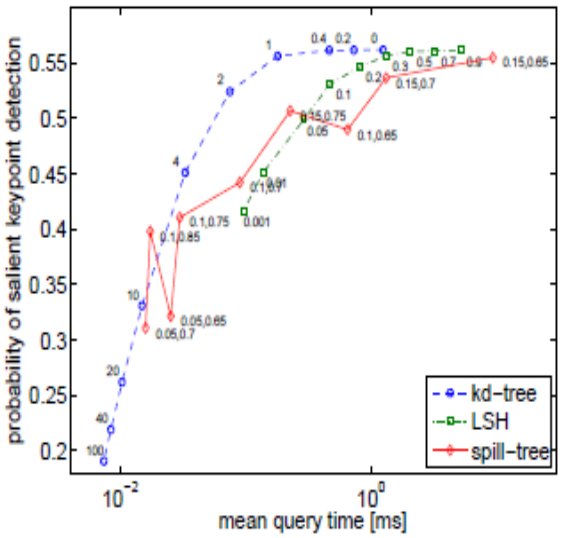

Figure2: Detection rate vs query time

The salient keypoint match detection rate for the three static kNN algorithms (because there are two parameters to vary in the case of spill-trees, we select a few key performance points in order to avoid cluttering the figure). Here, we see that the algorithms are able to achieve a maximum accuracy of around 55%, though kd-trees are able to do so almost an order of magnitude faster than LSH, which is itself almost an order of magnitude faster than spill-trees.

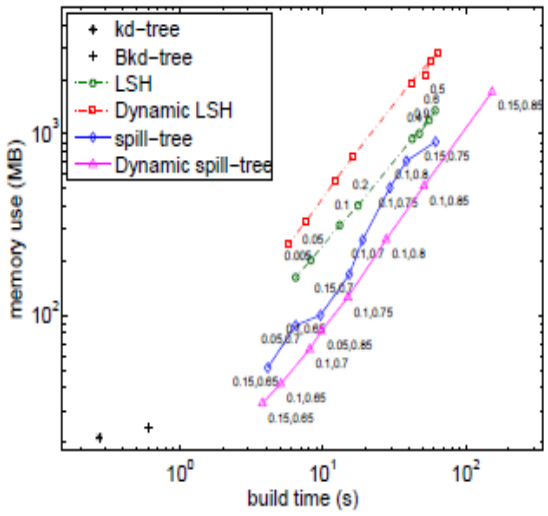

Figure 3 :Memory vs build time

While the query times are generally similar, when we additionally consider memory usage and data structure build times (Figure 2c) we can see that the kd-tree structure significantly outperforms both LSH and spill-trees and is therefore the method of choice for our application. Additionally, memory and build time are constant for the kd-tree, as the approximation parameter ϱ affects only the search algorithm, not the data structure.

6. CONCLUSION

In existing system, the problem of anonymizing sparse, highdimensional transactional data is solved through methods based on *(i)* local NN-search and *(ii)* global data reorganization. To handle well high data dimensionality, LSH-based anonymization outperforms in terms of data utility, but incurs slightly higher computational overhead. In this paper, KD-tree is proposed to replace the existing technique(LSH). kd-tree-based structures have the best performance in terms of accuracy, query time, build time, and memory usage. The primary contribution of this paper is demonstrating that building a NN search structure can be fruitfully viewed. A kd-tree is a data structure for storing a finite set of points from a k-dimensional space. A kd-tree is a binary tree. In KD-tree, search is depth first and uses the heuristic of searching first the child node which contains the target.

7. REFERENCES

[1] A.Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *ommunications of the ACM*, 51(1):117–122, January 2008.

[2] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. *Journal of the ACM*, 45: 891–923, 1998.

[3] G. Ghinita, Y. Tao, and P. Kalnis, "On the Anonymization of Sparse, High-Dimensional Data," in *Proc. of ICDE*, 2008, pp. 715–724.

[4] R. Agrawal and R. Srikant, "Privacy Preserving Data Mining," in *Proc. of ACM SIGMOD*, 2000, pp. 439–450.

[5] Mayur Datar, Nicole Immorlica, Piotr Indyk_," Locality-Sensitive Hashing Scheme Based on p-Stable Distributions".

[6] Piotr IndykA,"pproximate Proximity Problems

in High Dimensions via Locality-Sensitive Hashing", Helsinki, May 2007

[7] Lawrence Cayton, Sanjoy Dasgupta,"A Learning Framework for Nearest Neighbor Search".

[8] G. Ghinita, P. Karras, P. Kalnis, and N. Mamoulis, "Fast Data Anonymization with Low Information Loss," in *Proc. of VLDB*, 2007, pp.
758–769.