A computationally efficient and scalable approach for privacy preserving kNN classification

Sairam Ravu*§, P. R. Neelakandan*§, M. R. Gorai§, R. Mukkamala‡, P. K. Baruah §
§Sri Sathya Sai Institute of Higher Learning, Prashanthi Nilayam, India.
‡ Old Dominion University, Norfolk, Virginia, USA.
{ravusairam, neelaasai, mohangorai}@gmail.com, mukka@cs.odu.edu, pkbaruah@sssihl.edu.in

***Abstract*—In the modern age, there is a great desire to mine users' personal data from varied sources, to discover their behaviours. However, due to the growing awareness among the organizations regarding the privacy of user data and the strict privacy regulations of government, there is a growing resistance to share data directly with others. Encryption is used in the literature to achieve privacy preservation in data mining. Our technique is based on the application of Bloom filters on the sensitive data while still being able to perform collaborative data mining, in particular the kNN classification. In this work, we propose a parallel implementation on GPUs of the most time consuming part of the algorithm, i.e., the similarity computation of the Bloom filtered records based on the modified Jaccard metric and the classification of records. From our findings, we conclude that the proposed parallel implementation, apart from being cost effective, is highly scalable to accommodate huge data. The parallel implementation has an average speed up of 20 over serial implementation. Further, the speed up increases with increase in the size of the data set considered.**

***Index Terms*—Privacy, Data Mining, Bloomfilters, Modified Jaccard measure, kNN Classifier, GPU.**

I. Introduction

The unprecedented growth in web, storage, and processor technology has led to the increase in the number of paper-less organizations. Today, organizations such as banks, credit card companies, hospitals, and insurance companies, collect and store huge amounts of information electronically. Such big data can be useful only if some meaningful information can be extracted out of it. In particular, there is a desire to learn customer behaviour so as to improve the products and services that are offered by organizations as well as to detect any fraudulent activities. In addition, several third parties are also interested in mining data from multiple sources so as to derive meaningful information about entities such as people's spending behaviour and the spread of diseases.

*Student Author

Data mining has been the primary tool in discovering the behavioural patterns of the underlying entities. Typically, organizations outsource their data mining tasks to third-party providers. In such cases, organizations' data needs to be shared with the data miner. This type of sharing customer/company data with third-parties has a potential for privacy compromise. In other words, third-parties may know information about customers (for example) that are otherwise considered private. The need for privacy is often due to governmental regulations, business ethics, and individuals' interests.

Privacy Preserving Data Mining (PPDM) is the process of extracting knowledge from the data without compromising the privacy of confidential fields of the data. Over the recent years, to address the privacy concerns in data mining, researchers have introduced several schemes for mining data without compromising privacy. Among the many approaches that have been proposed in the past, encryption based techniques have been the most popular ones. Here, the sensitive attributes of the data are encrypted at the owner site prior to sending it to the data miner.

In this paper, we look at encrypting (or encoding) sensitive fields of data using Bloom filters. Some efforts in this direction, in the context of kNN classification, was reported in [1]. Bloom filters is a space-efficient probabilistic data structure, that is used to represent a set with support for membership queries [2] . kNN algorithm is a widely used method for classification in machine learning and pattern recognition. The method, though simple, is computation-intensive. This computational complexity arises since for each record in the input data, its similarity measure to all the records in the training set need to be computed. In the previous work, modified Jaccard metric [1] was used as a distance measure. We extend their work by parallelization of the distance computation step. In particular, in this work, we propose a parallel GPU-based CUDA implementation

of this algorithm. To the best of our knowledge, this work is first of its kind. The major contribution is the performance enhancement of the most time consuming part of the algorithm through GPUs.

The paper is organized as follows. In section 2, we give a brief summary of the related work. Section 3 describes the privacy preserving kNN classification algorithm. In section 4, the proposed privacy-preserving approach with Bloom filters implemented on GPUs is described. Section 5 illustrates the approach with an example. In section 6, we describe the experimental set up and code optimizations implemented in the code. In section 7, we discuss the results. Finally, section 8 summarizes the contributions of the paper and the scope for future work.

II. Related work

Ever since the advent of parallel programming paradigms, many parallel data mining algorithms have been explored. Parallel decision tree, parallel ARM, and parallel clustering are some of the noticeable ones. Even parallel kNN classification algorithm has been proposed. Garcia et al.[3] introduced a fast k-nearest neighbour algorithm and a faster kNN algorithm was proposed in [4]. All of them provided a faster algorithm on the basis of the euclidean metric. In this paper we present an approach to perform kNN classification in the context of privacy preservation. Privacy preservation is achieved using encoding means. We make use of Bloom filters to perform encoding. Classification is done on the Bloom filtered data based on the modified jaccard metric.

III. Privacy preserving kNN classification

We now briefly describe the methodology to achieve privacy preservation in kNN classification as proposed in [1]. Let us consider that there are m data owners and n data users. The data owners have the capability to perform kNN classification locally. Data users on the other hand require data classification. For instance, the data owners could be different recruiting organizations that have collected behavioral information of applicants, like qualifications, how well the applicant performs in certain positions after recruitment and so on. Users could be organizations that are wanting to screen their large pool of job applicants. In order to make an effective screening at low cost, the users intend to use the large data maintained at the data owners.

However, due to privacy concerns, instead of original tuples, the encoded tuples are shared. Encoding is achieved by means of Bloom filters. Bloom filter (BF) is a bit vector of a specified length. BF algorithm, given a data value, computes its associated hash functions on the data and produces Bloom filter for that value. To enforce anonymity between the data owners and the data users, an intermediate Semi-trusted Third Party (STTP) is introduced. The classification on the Bloom filtered tuples is based on modified jaccard (MJ) metric. The MJ metric is defined as the ratio of number of equal bits to the total number of bits.

It is reported that the single Bloom filter representation of the entire tuple provides better classification accuracy. For this reason, we have experimented with the single bloom filter representation of the entire record. To cater to the overall performance enhancement of the privacy preservation algorithm proposed in [1], we have modified the architecture of the system. The performance enhancement is in terms of delivering higher response rate to the data users, who require data classification. In the modified architecture proposed by us, the data owner acts as a cloud based data mining service provider. Thus data owner provides mining as a service. The data owner makes use of GPU to provide the much needed higher response rate. Figure 2 describes the modified architecture for a simple scenario, consisting of a single data owner, a single data user and a semi-trusted third party. We claim that the modified architecture can be extended to include more number of owners and users. In the following section we describe the CUDA implementation to achieve the above described performance enhancement.

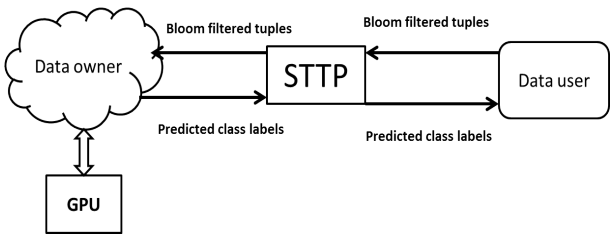

Fig. 1. Modified architecture for Privacy preserving kNN Classification

IV. Implementation of Privacy Preserving kNN classification on GPU using CUDA

The primary goal of our proposed implementation is to enhance the performance of kNN Classification algorithm on Bloom filtered data. In case of Bloom filtered data, classification is based on the modified jaccard (MJ) measure. Profiling results convey that the most time consuming part of the algorithm is the similarity

computation of the Bloom filtered records based on the modified Jaccard metric (taking 20% of the total execution time) and the determination of record classes (15 % of the total execution time). We propose a CUDA based implementation of the time consuming part on the GPU.

The approach is described as follows. The data owner, on receiving the Bloom filtered test records from the STTP, applies a faster GPU based kNN classification using the modified jaccard measure, considering the already existing Bloom filtered data as a training set. The steps involved in the faster similarity computation and class deduction using GPUs form the core of our contribution.

The following are the steps involved in achieving the high response rate from the data owners:

- Transfer of Bloom filtered test records, Bloom filtered training records and the associated class labels of the training records from the host to GPU.
- Launching the kernel "ComputeSimilarityMeasure" to compute similarity measure matrix.
- Sorting the similarity measure matrix, with respect to each test record, in the ascending order on the GPU.
- After sorting the similarity measure matrix, the following computations are achieved by the launch of another kernel "ComputeClasses".
  - Selecting the top k measures.
  - Identifying the class to which a test record belongs to based on the top k measures and associated class labels.

In the following, we provide the details of the above mentioned four steps.

1) Transfer of Bloom filtered records and the labels to GPU: In order to compute the similarity measures, the Bloom filtered records (both test and training) have to be transfered to the GPU. *Thrust* library provided by nVIDIA [5] , is used to perform this transfer. The advantage of using the *Thrust* library is that it makes common operations concise, efficient and readable.

2) Launching kernel to compute similarity measure matrix: The inherent parallelism in the computation of similarity measures is exploited by launching the kernel "ComputeSimilarityMeasure" with as many threads as the number of training records. The choice of having number of threads as that of number of training records is to make each thread responsible for one of the training records. The responsibility of each thread is as follows

- Computing MJ similarity measures between one training record and all the test records.
- Copy the class label of the assigned training record in a global array.

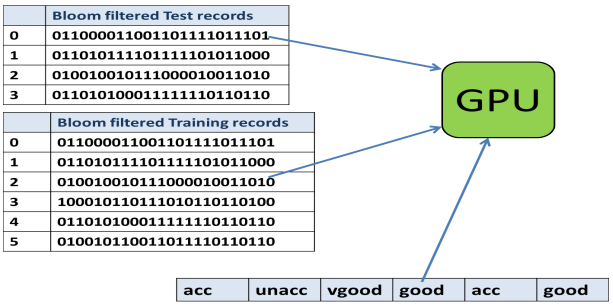

Fig. 2. Transfer of Bloom filtered records and labels on to the GPU

3) Sorting the similarity measure matrix, with respect to each test record in the ascending order: For each of the test records, the similarity measures obtained are considered as keys and the corresponding class labels of the training records are considered as values. Using the *Thrust* library sort_by_key the similarity measures with respect to each of the test records are sorted .

4) Launching kernel to deduce classes: The kernel "ComputeClasses" is launched with as many threads as the number of test records. The choice of having number of threads as that of number of test records is to make each thread responsible for deducing label for each of the test record.

V. Illustration

A. Transfer of Bloom filters to GPU

As shown in the Figure 2, Bloom filters and the labels are transfered to the GPU.

B. Launching kernel to compute similarity measure matrix

In the kernel, each thread is responsible for computing a column in the similarity matrix as shown in the Table I. The same thread is also responsible for copying the corresponding labels of those training records in the the labels matrix, as shown in the Table II. For example, thread 0 computes SimilarityMatrix[0][0] by comparing training record 0 and test record 0, computes SimilarityMatrix[1][0] by comparing training record 0 and test record 1 and so on. Also, thread 0 copies training record 0's label which is *acc* into all the entries in the column 0. Similarly, all the remaining threads perform the same procedure.

Thread-0	Thread-1	Thread-2	Thread-3	Thread-4	Thread-5
1	0.548	0.333	0.230	0.371	0.333
0.548	1	0.371	0.333	0.411	0.371
0.333	0.371	1	0.454	0.371	0.5
0.371	0.371	0.411	0.5	0.777	1

TABLE I
SIMILARITY MEASURES

Thread-0	Thread-1	Thread-2	Thread-3	Thread-4	Thread-5
acc	unacc	vgood	good	acc	good
acc	unacc	vgood	good	acc	good
acc	unacc	vgood	good	acc	good
acc	unacc	vgood	good	acc	good

TABLE II
LABELS

0.230	0.333	0.333	0.371	0.548	1
0.333	0.371	0.371	0.411	0.548	1
0.333	0.371	0.371	0.454	0.5	1
0.371	0.371	0.411	0.5	0.777	1

TABLE III
SIMILARITY MEASURES AFTER SORTING

good	vgood	good	unacc	acc	acc
good	vgood	good	acc	acc	unacc
acc	unacc	acc	good	good	vgood
acc	vgood	unacc	good	good	acc

TABLE IV
LABELS AFTER SORTING

C. Sorting the similarity measure, with respect to each test record in the ascending order

Now, each row in SimilarityMeasure matrix is sorted by using *Thrust* library's sort_by_key algorithm. So, as shown in the Table III, all the similarity measures are sorted in the ascending order and all the labels are moved to their corresponding slots as that of measures (Table IV).

D. Launching kernel to deduce classes

A kernel is launched with as many threads as that of number of test records, in the above example, four threads. Now, each thread finds the top k labels in each row. For example, if k is 3, then thread 0 goes to the end of first row, finds out that label *acc* has got the highest similarities measure which is 1.548 compared to that of *unacc's* 0.371 Hence, test record 0 is labeled *acc* by thread 0. Similarly, test records 1, 2 and 3 are labeled as *unacc*, *vgood* and *good* by threads 1, 2 and 3 respectively.

VI. EXPERIMENTAL SETUP

The experiments were conducted on Lonestar supercomputer. Lonestar provides NVIDIAs Tesla M2070 card with 448 cores and 6 GB global memory. The *Car* and the *Adult* data sets, available at UCI machine learning repository (http://archive.ics.uci.edu/ml/) were used.

Various optimizations are applied in our method to utilize the GPU efficiently. First, we employed memory coalescing—a technique in which threads accessing the neighbouring and sequential addresses in a coalesced manner. The proposed method will update the similarity matrix in a memory coalesced manner. The memory accesses of threads in a warp are coalesced into a single global memory transaction, rather than several transactions. Second, we sort a structure of arrays rather than an array of structures, since it is more efficient to do this way.

VII. RESULTS

In order to observe the efficacy of GPU implementation, we have duplicated the original records in *Car* dataset to obtain a data set of size 13640. The occupancy metric is defined as the ratio of number of active thread groups per processor to that of maximum number of thread groups per processor. Using the occupancy calculator provided by NVIDIA, we found that the thread block size of 480 provides optimal occupancy of 94%. Hence, kernel with thread block size of 480 was launched. Experiments were conducted for different k values (k-nearest neighbour). From our investigations (Table V), we found that an average speed up of 510 is achieved, considering only the computation time. Further, as the Table VI shows, an average speed up of 20 is achieved for the *Car* data set (considering both the memory transfer rate to the GPU and the computation time) with O0 compiler option. With O3 optimization an average speed up of 10 is achieved as shown in the Figure 3. This difference can be attributed to the compiler optimizations like loop unrolling and vectorization on the serial code.

We also have verified the scalability of our proposed method by experimenting on *Adult* data set. We found that an average speed up of 40 can be achieved with O0 compiler optimization (Table VII) and an average speed up of 20 with O3 optimization (Figure 4).

	k=3	k=5	k=9	k=15
Serial Code	21.90	21.89	22.05	22.06
GPU	0.04	0.04	0.04	0.04
Speedup	507.80	508.07	511.63	511.86

TABLE V
EXECUTION TIME (IN SECS) AND SPEEDUPS WHEN ONLY COMPUTATION IS TIMED FOR CAR DATA SET

	k=3	k=5	k=9	k=15
Serial Code	67.03	66.83	67.01	67.31
GPU	3.25	3.25	3.25	3.25
Speedup	20.62	20.539	20.61	20.70

TABLE VI
EXECUTION TIME (IN SECS) AND SPEEDUPS OF CAR DATA SET WHEN COMPILED WITH O0 OPTION

From our investigations, we observed that the speed up factor increases, with the increase in data size and that the parallel implementation always guarentees correctness.

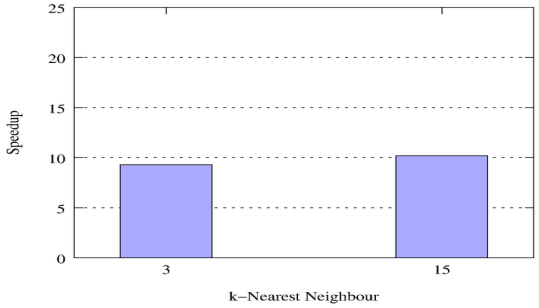

Fig. 3. Speed up obtained using GPU with Car data set compiled with O3 option

VIII. CONCLUSION AND FUTURE WORK

In this paper, we presented a CUDA-based parallel implementation of privacy preserving kNN classification.

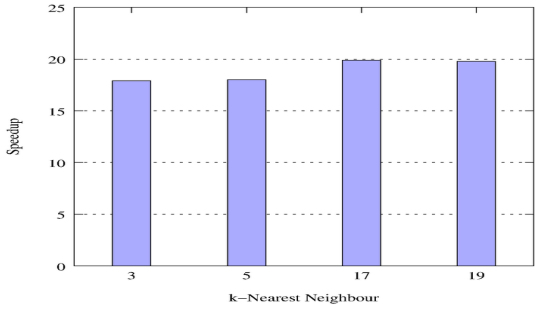

Fig. 4. Speed up obtained using GPU with Adult data set when compiled with O3 option

	k=3	k=5	k=9	k=15
Serial Code	483.00	483.50	484.50	485.40
GPU	11.86	11.86	11.88	11.85
Speedup	40.71	40.75	40.75	40.93

TABLE VII
EXECUTION TIME (IN SECS) AND SPEEDUPS OF ADULT DATA SET WHEN COMPILED WITH O0 OPTION

It is actually a hybrid implementation using the CPU and the GPU. Bloom filters are computed on CPU, similarity measures, sorting and identifying the labels is done on GPU. Data elements in these two kernels are processed in a data-parallel fashion. Optimizations such as memory coalescing techniques are used to reduce the number of memory write transactions onto the GPU global memory. Experiments showed good scalability on well known data sets. An average speedup of 20 over serial implementation was achieved. The results showed that our proposed method is a promising solution for privacy preserving kNN classification. One possible extension to this work could be to overlap the CPU-GPU communication and the computation on the GPU by using CUDA streams. Another could be to scale this approach to multiple GPUs.

ACKNOWLEDGMENT

We dedicate this work to Bhagawan Sri Sathya Sai Baba, Founder Chancellor of Sri Sathya Sai Institute of Higher Learning. This work was partially supported by nVIDIA, Pune, grant under Professor partnership program and the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number OCI-105375.

REFERENCES

[1] M. R. Gorai et al. Employing bloom filters for privacy preserving collaborative knn classification. *World Congress on Information and Communication Technologies(WICT)*, 2011.

[2] Bloom H. S. Space/time trade-offs in hash coding with allowable errors,. *Communications of the ACM, Volume 13, Issue 7, pp. 422-426*, 1970.

[3] V. Garcia et al. Fast k nearest neighbour search using gpu. *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2008.

[4] Shenshen Liang et al. Design and evaluation of a parallel k-nearest neighbor algorithm on cuda-enabled gpu,. *IEEE symposium on Web Society (IWS)*, 2010.

[5] Jared Hoberock and Nathan Bell. Thrust: A parallel template library. 2010.