International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Impact Factor (2012): 3.358

A Survey of Random Decision Tree Framework Privacy Preserving Data Mining

Vina M. Lomte[1], Hemlata B. Deorukhakar[2]

[1]Professor Computer Engineering Department, RMD Sinhgad College of Engineering, Pune, India

[2]M.E. Computer Engineering Department, RMD Sinhgad College of Engineering, Pune, India

Abstract*: Data mining with data privacy and data utility has been emerged to manage distributed data efficiently. In this paper, to deal with this advancement in privacy preserving data mining technology using accentuate approach of Random Decision Tree (RDT). Random Decision Tree provides better efficiency and data privacy than Cryptographic technique. Cryptographic technique is too slow and infeasible to enable truly large scale analytics to manage era of big data. Random Decision Tree is used for multiple data mining task like classification, regression, ranking, and multiple classifications. Privacy preserving RDT uses both randomization and cryptographic technique which provide data privacy for some decision tree based learning task.*

Keywords: Privacy preserving RDT, Data mining, Encryption, Decryption, Cryptography technique.

1. Introduction

Data Mining is Fast growing field of distributed Environment and process of discovering interesting patterns and knowledge from large database. It is also called as KDD process i.e. Knowledge Discovery from Data. It allows data analysis while preserving data privacy.

Data privacy preserving is prevent personal secret or private data from unnecessarily distributed or publicly known or not be misused by third person or by adversary. In privacy preserving data Mining, interesting and useful information is distributed with privacy of confidential information has been preserved. There are two stages in privacy preserving data Mining first is data collection and second data publishing. In data collection, data holder stores data which is gathered by data owner. In data publishing, data can be released to data recipient by data holder and data recipient mines published secured data.

Cryptographic techniques are often too slow to be practical and can become computationally expensive as the rise in size of the data set and communications between various parties increase [1]. Cryptographic techniques cannot handle big data. In this paper, we are using privacy preserving RDT is Random Decision Tree with privacy preserving data mining which is developed by Fan et al. [3]. Privacy preserving RDT is combination of randomization and cryptography technique.

This solution provides an order of magnitude improvement in efficiency over existing solutions while providing more data privacy and data utility. This is an effective solution to privacy-preserving data mining for the big data challenge. Random Decision Tree provides better efficiency and data privacy than Cryptographic technique. RDT provides a structural property, more specifically, the fact that only specific nodes (the leaves) in the classification tree need to be encrypted/decrypted, and secure token passing prevents adversary from utilizing counting techniques to decipher instance classifications, as the branch structure of the tree is hidden from all parties. RDT to generate trees that are random in structure, providing us with a similar end effect as perturbation without the associated pitfalls. A random structure provides security against leveraging a priori information to discover the entire classification model or instances.

2. Random Decision Tree

Random decision tree algorithm constructs multiple iso-depth decision trees randomly. RDT is based on two stages, training and classification and s structure of a random tree is constructed completely independent of the training data. When constructing each tree, first, start with a list of attributes from the data set. Generate a tree by randomly choosing one of the attributes without using any training data. The tree stops growing once the height limit is reached. Then, use the training data to update the statistics of each node. In this only the leaf nodes need to record the number of values of different classes that are classified through the nodes in the tree. The training data is scanned exactly once to update the statistics in multiple random trees. When classifying a new instance j, the probability outputs from multiple trees are averaged to estimate the a posteriori probability.

The training phase consists of creating the trees (BuildTreeStructure) and populating the nodes with training instance data (UpdateStatistics). It is assumed that the number of attributes is known to all parties based on the training data set. The depth of each tree is decided based on a heuristic Fan et al. [3] show that when the depth of the tree is equal to half of the total number of attributes present in the data, the most diversity is achieved, preserving the advantage of random modeling.

In RDT, tree stops growing any deeper if one of the following conditions is met:

- A node becomes empty or there are no more examples to split in the current node.
- The depth of tree exceeds some limits.

Each leaf node of the tree records class distributions. Assume that a leaf node has a total of 1000 examples from the training data that pass through it. Among these 1000 examples, 200 of them are + and 800 of them are -. Then the class probability distribution for + is P (+|x) = 200/1000 = 0.2 and the class probability distribution for - is P (-|x) = 800/1000 = 0.8

Large-Scale distributed applications are subject to frequent disruptions due to resource contention and failure. Such disruptions are unpredictable and therefore robustness is a designable property for the distributed operated environment. Describe and evaluate a robust topology for applications that operate on a Random tree overlay network. This technique is used for improving the robustness of a distributed system. Random trees are used in communication network to disseminate information from one node to all other nodes and/or to collect information at a single designated node. The most common Random trees are shortest path and minimum Random tree.

3. Example

Table 1 shows the weather data set distributed between two different parties. In this first, data set is horizontally partitioned; instances 1-7 are owned by Party 1, while 8-14 are owned by Party 2. If it is vertically partitioned, Party 1 owns the outlook and temperatures attributes while Party 2 owns the humidity, windy, and play attributes. Suppose a new instance {sunny, mild, normal, and weak} is to be classified. Then, as per the first random tree, the prediction is (2, 0) without normalization. The prediction as per the second random tree is (1, 2). Therefore, the non-normalized overall class distribution vector provided by RDT is (1.5, 1).

Table 1: The Distributed Weather Data Set

	—P1—		—P2—		
	outlook	temperature	humidity	windy	play
P1	sunny	hot	high	weak	no
	sunny	hot	high	strong	no
	overcast	hot	high	weak	yes
	rainy	mild	high	weak	yes
	rainy	cool	normal	weak	yes
	rainy	cool	normal	strong	no
	overcast	cool	normal	strong	yes
P2	sunny	mild	high	weak	no
	sunny	cool	normal	weak	yes
	rainy	mild	normal	weak	yes
	sunny	mild	normal	strong	yes
	overcast	mild	high	strong	yes
	overcast	hot	normal	weak	yes
	rainy	mild	high	strong	no

4. Problem Statement

RDT code can be used for multiple data mining tasks, basic problem in distributed classification is to train a classifier from the distributed data and then classify each new instance. For distributed random decision tree classification, the objective is to create a random decision tree classifier from the distributed data. In the privacy-preserving case, the additional constraint is that the process of building the classifier or of classifying an instance should not leak any additional information or personal information beyond what is learned from the result (and the local input). Assuming the global data set D ≡ (T, R), where T represents the global set of transactions, and R represents the global schema, the general problem can be formulated as follows.

Definition 1: **(Privacy-Preserving RDT). Given a data set D ≡ (T,R) distributed among k parties P_1; . . . ; P_k, securely build a random decision tree classifier RDT, and provide a privacy- preserving distributed classification mechanism to classify a new instance.**

There are two dependent step to data is distributed. Horizontal and Vertical partitions. In this, when data is horizontally partitioned between P_k parties, each party holds different instances, but collects the same section of information. All parties share the schema, though the specific transactions in their local databases are unique. Clearly, since the schema is shared by all parties, the class attribute C is also known to all parties.

5. Horizontally Partition Data

In this paper, when data is horizontally partitioned between P_k parties, each party holds different instances, but collects the same section of data. Each party can only independently create the structure of the tree. All parties must co-operatively and securely compute the parameters over global dataset. There are two options: 1) The structure of the tree is known to each party. 2) The structure of the tree is unknown to each party. Algorithm 1 gives the details.

Algorithm 1 Building the random trees for horizontally partitioned data

Require: Transaction set T partitioned horizontally between sites $P_1, \ldots, P_k$
Require: n_i, the number of random trees to be created by each participant, such that $\sum_i n_i = m$, the total number of random trees
1: **while** Every party does not agree to all of the random trees **do**
2: {A secure electronic voting protocol can be used if no party should learn which party objects to any of the trees}
3: Each party generates its random trees
4: The structure of every tree is communicated to all of the parties
5: **end while**
6: **for** each tree T_j **do**
7: Each party P_i locally computes the class distribution vectors for each leaf node in T_j
8: Each party P_i encrypts the class distribution vectors for all leaf nodes in T_j using the threshold additively homomorphic encryption system and sends to all other parties
9: All parties then multiply the corresponding encrypted class distribution vector elements they receive for each leaf node to get the encrypted global value for that node
10: **end for**

When a new instance needs to be classified, the party owning the instance identifies all of the leaf nodes that it reaches, and multiplies the encrypted class vector components together to get the encrypted sum of the class distribution vectors as per each tree. This is now collaboratively decrypted and averaged to get the actual class distribution vector. Note that, getting the sum does not reveal more than getting the average since the sum can always be retrieved from the average and the total number of trees. Algorithm 2 gives the details.

6. Vertically Partition Data

Vertically partitioned data, all parties collect data for the same set of values. Each party collects data for a different set of attributes. Parties cannot independently create even the structure of a random tree, unless they share the attribute information among each other. Thus, there are two options: All parties share basic attribute information so they can independently create random trees (at least the structure). There is no sharing of information. Now, the parties need to collaborate to create the random trees. These trees could themselves exist in a distributed form.

6.1 Fully Distributed Trees

Unlike the horizontal partitioning case, the structure of the tree does reveal potentially sensitive information, since the parties do not know what the attributes are owned by the other parties. Therefore, we directly address the case of fully distributed trees. Every site knows the total number of random trees, denoted by *m*. The algorithms for creating random trees are given in Algorithm 3.

Algorithm 2 classifyInstanceHoriz(instId)

Require: m, the number of random trees built (let $nodeid_j$ represent the root node of the jth tree)
1: **for** $i = 1 \ldots p$ **do**
2: Randomly choose r from the range of random numbers for the cryptographic system
3: $lv_i \leftarrow Encrypt(0, r)$ {Random encryption of 0}
4: **end for**
5: **for** $j = 1 \ldots m$ **do**
6: $currstats \leftarrow$ encrypted $classdistributionvector$ for instance $instId$ as per tree j
7: **for** $i = 1 \ldots p$ **do**
8: $lv_i \leftarrow lv_i * currstats_i$
9: **end for**
10: **end for**
11: Collaboratively decrypt each lv_i and divide by m to get actual statistics

Algorithm 3 CreateRandomTrees

Require: Transaction set T partitioned vertically between sites $P_1, \ldots, P_k$
Require: P_i holds n_i attributes
Require: p class values, $c_1, \ldots, c_p$, with P_k holding the class attribute
Require: m, the number of random trees to build
1: All parties together compute $n = \sum_i n_i$ using the secure sum protocol [10]
2: $depth \leftarrow n/2$ {The depth of the random trees}
3: **for** $i = 1 \ldots m$ {Build the ith tree} **do**
4: $level \leftarrow 1$
5: $nodeId_i \leftarrow BuildTree(level, depth)$
6: **end for**

Algorithm 3 that invokes Build Tree Algorithm 4 for *m* times, in order to build *m* random trees of depth n/2, where n is the total number of attributes in the global schema R. depth of random tree is half of the total number of attributes and is computed by all sites together using the secure sum protocol [2]. Additively homomorphic encryption is used for computation of the sum of encrypted values for updating the class statistics in Algorithm 5. Also, random encryption prevents disclosure of which class value is updated during statistics updating phase.

Algorithm 4 BuildTree(level,depth)

if $level \leq depth$ **then**
  Randomly choose j from $[1 \ldots k]$ (uniform random distribution)
  At P_j: Randomly choose r from $[1 \ldots n_j]$ (uniform random distribution)
  At P_j: Create Interior Node Nd with attribute $Nd.A \leftarrow A_r$ (the rth attribute)
  if A_r is numeric **then**
    At P_j: Randomly choose a split point π from within the range of A_r
    At P_j: $Constraints.set(Nd.A_r, \text{“} < \pi \text{”})$ {Update local constraints tuple}
    $nodeId \leftarrow BuildTree(level + 1, depth)$
    $Nd.leftsubtree \leftarrow nodeId$ {Add appropriate branch to interior node}
    At P_j: $Constraints.set(Nd.A_r, \text{“} >= \pi \text{”})$ {Update local constraints tuple}
    $nodeId \leftarrow BuildTree(level + 1, depth)$
    $Nd.rightsubtree \leftarrow nodeId$ {Add appropriate branch to interior node}
  else
    for each attribute value $a_i \in Nd.A$ **do**
      $Constraints.set(Nd.A_r, a_i)$ {Update local constraints tuple}
      $nodeId \leftarrow BuildTree(level + 1, depth)$
      $Nd.a_i \leftarrow nodeId$ {Add appropriate branch to interior node}
    end for
  end if
  $Constraints.set(A_r, \text{'?'})$ {Returning to parent: should no longer filter transactions with A_r}
  Store Nd locally keyed by Node ID
  return Node ID of interior node Nd {Execution continues at site owning parent node}
else
  for $i = 1 \ldots p$ **do**
    Randomly choose r from the range of random numbers for the cryptographic system
    $lv_i \leftarrow Encrypt(0, r)$ {Random encryption of 0}
  end for
  Create leaf node Nd with p values $lv_1, \ldots, lv_p$
  return Node ID of leaf node Nd
end if

6.1.1 Update Statistics

For updating the statistics, RDT algorithm, for each random tree and instance in the training dataset, the tree is traversed to the appropriate leaf node. At the leaf node, the element in the class distribution vector corresponding to the class label of the instance is incremented by one. This process is repeated for all the random trees. In the vertically partitioned data, the nodes and the attributes of the dataset are distributed across multiple sites. While updating statistics no site should learn the attributes and attribute values in the dataset of other sites. The algorithm for update statistics for vertically partitioned data is given in Algorithm 5 that calls increment Stats for distributed tree traversal and incrementing the class statistics using additively homomorphic encryption for preventing information leakage. Algorithm 6 is gives the details.

Algorithm 5 UpdateStatistics()

Require: Transaction set T partitioned vertically between sites $P_1, \ldots, P_k$
Require: P_u holds n_u attributes
Require: p class values, $c_1, \ldots, c_p$, with P_k holding the class attribute
Require: m, the number of random trees built (let $nodeid_j$ represent the root node of the jth tree)

for each instance t_x **do**
  {At P_k, build the class vector}
  for $i = 1 \ldots p$ **do**
    Randomly choose r from the set of positive integers
    if t_x has class c_i **then**
      $lv_i \leftarrow Encrypt(1, r)$ {Random encryption of 1}
    else
      $lv_i \leftarrow Encrypt(0, r)$ {Random encryption of 0}
    end if
  end for
  for $j = 1 \ldots m$ **do**
    $incrementStats(t_x, nodeId_j, lv)$
  end for
end for

Algorithm 6 incrementStats(instId, nodeId, lv): increments the class/distribution for the instance represented by instId

{The start site and ID of the root node is known}
if $nodeId$ is a LeafNode **then**
  {update leaf values in $nodeId$}
  for $i = 1 \ldots p$ **do**
    $nodeId.val_i \leftarrow nodeId.val_i * lv_i$ {Increment by 1 or hide}
  end for
else
  {$nodeId$ is an interior node}
  $Nd \leftarrow$ local node with id $nodeId$
  if Nd.A is numeric **then**
    if value of attribute $Nd.A$ for transaction $instId$ is smaller than split point **then**
      $childId \leftarrow Nd.leftsubtree$
    else
      $childId \leftarrow Nd.rightsubtree$
    end if
  else
    $value \leftarrow$ the value of attribute $Nd.A$ for transaction $instId$
    $childId \leftarrow Nd.value$
  end if
  childId.Site.$incrementStats(instId, childId, lv)$ {Actually tail recursion: this site need never learn the class}
end if

6.1.2 Performing Classification

Instance classification also proceeds in a distributed fashion similar to update statistics. For instance classification, the prediction of the given instance is computed by taking an average of the probability outputs from multiple RDTs. The distributed algorithm for instance classification is given in Algorithm 7. Then return these class/ distribution for instance the instance represented by instID in tree given by nodeID.

Algorithm 7 classifyInstance(instId)

Require: m, the number of random trees built (let $nodeid_j$ represent the root node of the jth tree)

1: **for** $i = 1 \ldots p$ **do**
2: Randomly choose r from the range of random numbers for the cryptographic system
3: $lv_i \leftarrow Encrypt(0, r)$ {Random encryption of 0}
4: **end for**
5: **for** $j = 1 \ldots m$ **do**
6: $currstats \leftarrow retriveStats(instId, nodeId_j)$
7: **for** $i = 1 \ldots p$ **do**
8: $lv_i \leftarrow lv_i * currstats_i$
9: **end for**
10: **end for**
11: Collaboratively decrypt each lv_i and divide by m to get actual statistics

7. Conclusion and Future Work

In this paper, we studied the technical feasibility of realizing privacy-preserving data mining. RDTs can be used to generate equivalent, accurate and sometimes better models with much smaller cost; we are using distributed privacy-preserving RDTs. Our approach leverages the fact that randomness in structure can provide strong privacy with less computation. In the future, we plan to develop general solutions that can work for arbitrarily partitioned data and overlapping transaction.

References

[1] Jaideep Vaidya, Senior Member, IEEE, Basit Shafiq, Member, IEEE, Wei Fan, Member, IEEE, Danish Mehmood, And David Lorenzi "A Random Decision Tree Framework Or Privacy-Preserving Data Mining" Proc. IEEE Transactions On Dependable And Secure Computing, Vol. 11, No. 5, September/October 2014
[2] J. Vaidya, C. Clifton, and M. Zhu, Privacy-Preserving Data Mining.ser. Advances in Information Security first ed., vol. 19, Springer-Verlag, 2005.
[3] W. Fan, H. Wang, P.S. Yu, and S. Ma, "Is Random Model Better? On Its Accuracy and Efficiency," Proc. Third IEEE Int'l Conf. Data Mining (ICDM '03), pp. 51-58, 2003.
[4] W. Fan, J. McCloskey, and P. S. Yu, "A General Framework for Accurate and Fast Regression by Data Summarization in Random Decision Trees," Proc. 12th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD '06), pp. 136-146, 2006.
[5] X. Zhang, Q. Yuan, S. Zhao, W. Fan, W. Zheng, and Z. Wang,"Multi-Label Classification without the Multi-Label Cost," Proc. SIAM Int'l Conf. Data Mining (SDM '10), pp. 778-789, 2010.
[6] A. Dhurandhar and A. Dobra, "Probabilistic Characterization of Random Decision Trees," J. Machine Learning Research, vol. 9, pp. 2321-2348, 2008.
[7] G. Jagannathan, K. Pillaipakkamnatt, and R.N. Wright, "A Practical Differentially Private Random Decision Tree Classifier," Proc. IEEE Int'l Conf. Data Mining Workshops (ICDMW '09), pp. 114-121,2009.
[8] J. Vaidya, C. Clifton, M. Kantarcioglu, and A.S. Patterson," Privacy-Preserving Decision Trees over Vertically Partitioned Data," ACM Trans. Knowledge Discovery from Data, vol. 2, no. 3,pp. 1-27, 2008.
[9] O. Goldreich, "General Cryptographic Protocols," The Foundations of Cryptography, vol. 2, pp. 599-764, Cambridge Univ. Press, 2004.

Author Profile

Prof. Vina M. Lomte, received the B.E. degree in Computer Science and Engineering from Amravati University, BNCOE, Pusad and M.E. degree in Computer Engineering from Mumbai University, MGMCET, Kamothe. Currently working as Assistant Professor of Computer Engineering Department in RMD SSOE Pune, Maharashtra, India.

Hemlata B. Deorukhakar received the B.Tech. Degree in Computer Science and Engineering from IGNOU, New Delhi in 2013. Currently appearing M.E. 2nd year Computer Engineering in RMD SSOE Pune, Maharashtra, India