Efficient Privacy Preserving Nearest Neighboring Classification from Tree Structures and Secret Sharing

Jhe-Kai Yang, Kuan-Chun Huang, Cheng-Yang Chung, Yu-Chi Chen, and Ting-Wei Wu
Department of Computer Science and Engineering
Yuan Ze University
Taoyuan, Taiwan
{s1106002,s1096007,s1061453}@mail.yzu.edu.tw,
{wycchen,wtw}@saturn.yzu.edu.tw

***Abstract*—The k-nearest neighbor (kNN) algorithm is a very simple manner in the area of machine learning. It is a supervised method to classify according to the distance between different instances and is also widely used in solving some classification problems. It is expected to obtain better training with a larger dataset. However, how to perform kNN algorithm efficiently is an issue with privacy-preserving. In this paper, we proposed a privacy-preserving k-nearest neighboring scheme by secret sharing and improve the kNN classification by preprocessing with tree structures. Finally, the applicability of our method is shown by experiments with real datasets.**

***Index Terms*—Privacy, Nearest neighbor algorithm, Secret sharing, Tree structures**

I. Introduction

With the vigorous development of data science and machine learning technology, data becomes a very important asset. The utilization of relevant data is necessary for many organizations, which increases the opportunities for sharing and integrating data. In the traditional approach, we will collect data on the server side before analyzing it. When people provide data to the server, they are worried about their privacy will be endangered due to the leakage of the provided information. Therefore, the importance of protecting data privacy has been paid more attention, and there are some papers [1]–[4] focus on how to collect and integrate data while privacy-preserving.

The k-nearest neighbor (kNN) algorithm is a simple supervised learning method that classifies unknown things from a given dataset. The judgment method is to compute the distance with training instances (such as Euclidean distance, Manhattan distance, Chebyshev distance, etc.), finding the closest group of data, and determine the category by distinguishing which feature is more. By finding the nearest neighbors and using the majority label to determine the classification result, kNN has the advantage of being easy to comprehend and implement, which makes kNN widely used in many aspects, such as text categorization [5] and medical service [6].

The issue of data privacy leakage in machine learning has expressed great concern in [7]–[9]. Existing solutions use secure multi-party computing or homomorphic encryption to protect against privacy leaks. However, there are still significant challenges to balance security and efficiency. Particularly, in kNN algorithm, each classification requires calculation on the entire dataset, which means that a large dataset causes more operation on ciphertext while privacy-preserving. Thus, there is a problem of how to efficiently computes kNN algorithm on a large dataset while maintaining privacy.

There are some works focusing on efficient secure computation between kNN query and classification. Kantarcıoğlu et al. [10] presented a kNN classification on distributed databases, which is able to find the majority class from a global kNN consists of the local nearest results. However, the communication and computation cost is not cheap in the distributed sites situation. Rong et al. [11] proposed the outsourced collaborative kNN (OCkNN) protocol, which allows data owners to use their own keys for encryption, and perform joint kNN classification over the distributed encrypted database on clouds. Nevertheless, OCkNN is not rapid enough for large-scale datasets. Park et al. [12] designed a faster privacy-preserving protocol to find the top k-th (PE-FTK) highest similarity by comparing one bit at a time. Although the computational burden can be reduced with a shorter comparison, they have ignored the major computing cost in $computeSimilarity$ before PE-FTK. Sun et al. [13] developed a privacy-preserving Euclidean distance protocol (PPEDP), which can run faster on high-dimensional features sample set. Unfortunately, this formation has a shortcoming of the sharply increasing overhead when the sample set has more samples than features.

From the previous works, we can conclude that the amount of calculation and computing power of ciphertext is an important issue that cannot be underestimated. Since secret sharing can simply distributes and computes data by the shares of the secrets, this property make it suitable for secure multi-party computation. In this paper, we use secret sharing to construct the privacy-preserving kNN classification.

A. Our Contributions

In this article, we propose a novel privacy-preserving kNN (PPkNN) classification scheme from tree structures and secret sharing. The main contributions of this article are summarized as follows.

1) We design a series of protocols based on secret sharing, which can securely and efficiently calculate data between different parties. Compare with the existing solutions [10]–[13], our newly designed ones can save more computation time while protecting against privacy leaks.
2) Our solution maintains a certain accuracy of the traditional kNN while reducing the computational and communication burden by calculating the dataset partition from a decision tree structure.
3) We test the performance of our scheme and show the feasibility with some real-world datasets. In the experimental results, we also demonstrate that our scheme is extremely efficient for privacy-preserving nearest neighboring search. Specifically, PPkNN takes about 6 hours among 12960 instances, our scheme only needs less than half an hour on average.

B. Organization

This article will include the following content in the follow-up: In Section II, we introduce some preliminaries which will be used in this work. Section III mentions our system model and design goals. Section IV describes the structure we propose. Section V will talk about the method we use to improve the construction. Section VI shows the experimental results. Finally, the conclusions of this paper are given in Section VII.

II. Preliminaries

A. k-nearest Neighbor Algorithm

The k-nearest neighbor (kNN) algorithm is a case-based learning algorithm that classifies according to the distance between different training instances. The main purpose of the algorithm is to determine the classification result according to the majority of labels of k nearest neighbors.

k-nearest neighbor algorithm:

1. Preparing a training dataset of the labelled categories.
2. Using the Euclidean distance algorithm to calculate the distance between the test instance and all other instances in the training set.
3. Setting the value of k to find the k closest instances in the calculated distance as neighbors of the test instance.
4. From the k neighbors found, find the largest number of the categories, and judge the test instance as this category.

B. Secret Sharing

Secret sharing was first introduced by Shamir [14]. In this paper, we focus on 2-out-of-2 secret sharing under a prime modulo q. Let s denote to the secret, and the share of the secret is represented as $[s]$. We present a secret sharing scheme below.

- Share function can create two shares from the secret. $Share(s) \rightarrow [s]_1, [s]_2$.
- Recover function must use the two shares to rebuild the secret. $Recover([s]_1, [s]_2) \rightarrow s$.

The secret sharing method is also in line with the property of the addition homomorphism, suppose that there are two secrets x, y, and we let z be the result of the secrets addition, then $[z]_1 = [x]_1 + [y]_1$ and $[z]_2 = [x]_2 + [y]_2$. As the characteristic of secret sharing shows above, we can get that $z = [z]_1 + [z]_2 = [x]_1 + [y]_1 + [x]_2 + [y]_2$. With the property of homomorphic addition, it can be extended to do the addition and multiplication of the shares, and also compare secrets through shares.

III. System Definition

A. System Model

Our construction consists of three entities, including the server, client, and the randomness generator.

1) Server: In our system, the server is the owner of the dataset D. It is expected to provide kNN classification services without leaking the dataset.
2) Client: The client is the provider that proposes the query data Q. Before sending the query data to the server, the client will protect privacy through secret sharing. Then it can securely compare the data and receive the classification result safely from the server.
3) Randomness Generator: The randomness generator is a trusted third-party unit, which is responsible for providing randomly generated shares for the client and server to utilize.

B. Threat Model

In this paper, we assume that both the client and server are semi-honest (i.e. they honestly follow the construction but attempt to learn more information during processing). Moreover, we consider an adversary $\mathcal{A}$ whose goal is to obtain the dataset, query data, or classification results. $\mathcal{A}$ has the ability to eavesdrop on all communications and get the transmitting data between client and server.

IV. Naive Construction

In the data format of dataset $D = \{d_1, d_2, ..., d_n\}$, all data $d_i = \{v_{i,1}, v_{i,2}, ..., v_{i,m}, l_i\}$ is an instance including m-th attribute values and a category label l_i. We assume that the data is composed of rounding the floating-point numbers into integers, so the modulo calculation always holds. We expect the client protect their query data $Q = \{q_1, q_2, ..., q_t\}$ through secret sharing to increase the security of data privacy. Moreover, we take advantage of it to achieve secure multi-party computing, which communicate with the server to perform the classification of the kNN algorithm. It allows the server to complete analysis without knowing data in the plain.

A. Secret Sharing-based Building Blocks

Here, we use secret sharing to protect data privacy and ensure that none of the data information will be leaked. In the following, some building blocks are formally introduced, and later will be used to build our construction.

1) Randomness Generator: We rely on the randomness generator (RG) protocol, which can even randomly generate three groups of shares. This architecture is composed of three units, and party P_3 represents our Randomness Generator, party P_1 and party P_2 represent the unit of receiving random shares. It can generate two values a and b in a random manner, and use the a, b, ab through secret sharing to generate three groups of shares $([a]_1[a]_2, [b]_1[b]_2, [ab]_1[ab]_2)$. Then $[a]_1, [b]_1, [ab]_1$ is delivered to party P_1; $[a]_2, [b]_2, [ab]_2$ is transmitted to party P_2.

2) Data Sharing via Secret Sharing Multiplication: We design the secure secret sharing multiplication (SSSM) protocol with randomness generator (as shown in Protocol 1). This protocol can help us calculate the result of secret multiplication through share without revealing any information about secrets.

Protocol 1 Secure Secret Sharing Multiplication (SSSM)

P_1 **Input**: $[x]_1$,$[y]_1$
P_2 **Input**: $[x]_2$,$[y]_2$
P_1 **Output**: $[xy]_1$
P_2 **Output**: $[xy]_2$

1. P_1 and P_2 gets $[a]_1$, $[b]_1$, $[ab]_1$ and $[a]_2$, $[b]_2$, $[ab]_2$, respectively, from Randomness Generator.
2. P_1 sends $[x]_1 - [a]_1$ to P_2 and P_2 sends $[x]_2 - [a]_2$ to P_1 then gets $[x] - [a]$ to obtain $\varepsilon = x - a$.
3. P_1 sends $[y]_1 - [b]_1$ to P_2 and P_2 sends $[y]_2 - [b]_2$ to P_1 then gets $[y] - [b]$ to obtain $\rho = y - b$.
4. P_1 locally computes $[xy]_1 = [ab]_1 + \varepsilon[b]_1 + \rho[a]_1 + \varepsilon\rho$.
5. P_2 locally computes $[xy]_2 = [ab]_2 + \varepsilon[b]_2 + \rho[a]_2 + \varepsilon\rho$.

Protocol 2 Secure Comparison (SC)

P_1 **Input**: $[x]_1$,$[y]_1$
P_2 **Input**: $[x]_2$,$[y]_2$
P_1, P_2 **Output**: 1 if $(x > y)$ or 0 if $(x \leq y)$

1. P_1 or P_2 produces $[\ell]_1, [\ell]_2$, where $x, y \leq \ell$ (ℓ is the public parameter).
2. P_1 generates $[z]_1 = [x - y + \ell]_1$, and P_2 does $[z]_2$ = $[x - y + \ell]_2$, respectively.
3. P_1 uniformly picks two positive integers r and $r^{'}$, where $2\ell r + r^{'} < q$, and then generates $[r]_1, [r]_2, [r^{'}]_1, [r^{'}]_2$.
4. P_1 computes $[s]_1 = [z * r + r^{'}]_1$ and $[h]_1 = [\ell * r + r^{'}]_1$ and P_2 does $[s]_2 = [z * r + r^{'}]_2$ and $[h]_2 = [\ell * r + r^{'}]_2$ by applying Protocol 1.
5. P_1 sends $[s]_1$ and $[h]_1$ to P_2.
6. P_2 reconstructs s and h. P_2 outputs 1 if $s > h$, or 0 if not. (P_2 will inform the result to P_1.)

3) Secure Comparison of Secret Sharing: We designed the secure comparison (SC) protocol (as shown in Protocol 2). This architecture requires the assistance of the secure secret sharing multiplication protocol. Through the SC protocol, we can compare the relationship between secrets without leaking any information of their secret.

For party P_1 and party P_2 participating in the SC protocol, no party can obtain other information except the information implied by party P_1.

B. Proposed Protocol

1) Initialization: Our architecture is a three-party system, namely client, server, and randomness generator. Both the client and the server have their own secret s_1, s_2. First, we must locally convert their secret into the form of share. The client will use its secret s_1 to generate share $([x]_{1,1}, [y]_{1,1})$ through the equation $y = r_1 x + s_1$; the server will use its own secret s_2 to generate share $([x]_{2,1}, [y]_{2,1})$ through the equation $y = r_2 x + s_2$, which r_1, r_2 are generated by pseudo random number generator.

When two shares are produced, in order to meet the needs of subsequent calculations, we fix the value of parameter $[x]_{1,1}$ and $[x]_{2,1}$ and bring them to the other equation. Specifically, we use the parameter $[x]_{1,1}$ as $[x]_{2,2}$ in $y = r_2 x + s_2$ and get $[y]_{2,2}$. Similarly, we fix the parameter $[x]_{2,1}$ to $y = r_1 x + s_1$ and get $([x]_{1,2}, [y]_{1,2})$, where $[x]_{1,2}$ equals to $[x]_{2,1}$. Thus, we can compute on shares and ensure that the secret information will not be disclosed.

After the server and the client get their own shares, they will exchange one of the shares to the other. The client sent $([x]_{1,1}, [y]_{1,1})$ to the server and gets $([x]_{2,2}, [y]_{2,2})$; the server sent $([x]_{2,2}, [y]_{2,2})$ to the client and gets $([x]_{1,1}, [y]_{1,1})$.

Protocol 3 Positive Difference (PD)

P_1 **Input**: $[x]_1$,$[y]_1$
P_2 **Input**: $[x]_2$,$[y]_2$
P_1 **Output**: $[x - y]_1$ if $(x > y)$ or $[y - x]_1$ if $(x \leq y)$
P_2 **Output**: $[x - y]_2$ if $(x > y)$ or $[y - x]_2$ if $(x \leq y)$

1. P_1 and P_2 obtain the comparison result $c \leftarrow SC(([x]_1, [y]_1), ([x]_2, [y]_2))$ by applying Protocol 2.
2. P_2 outputs $[x]_2 - [y]_2$ if $c = 1$, or $[y]_2 - [x]_2$ if $c = 0$.
3. P_1 outputs $[x]_1 - [y]_1$ if $c = 1$, or $[y]_1 - [x]_1$ if $c = 0$.

2) Share Difference: The secure secret sharing multiplication protocol (Protocol 1) can imply the shares difference with some randomness. Then, we will utilize the secure comparison protocol (Protocol 2) to compare the shares and receive the relationship between the two secrets. With the relationship we can get the positive difference $\triangle$ by the positive difference protocol (Protocol 3) without knowing the secrets. Note that all calculations need to go through modulo calculations.

3) PPkNN Classifier: We can aggregate the differences $\triangle$ between the secrets as the distance data. Once we have the distance between instances and queries, we can securely sort them on shares with secure sort protocol (Protocol 4).

Then find the k nearest data to determine which category the query data belongs to. The complete privacy-preserving kNN is shown in the main protocol (Protocol 5)

Protocol 4 Secure Sort (SS)

P_1 **Input**: $[V]_1 = \{[v_1]_1, [v_2]_1, ..., [v_n]_1\}$, $L = \{l_1, l_2, ..., l_n\}$
P_2 **Input**: $[V]_2 = \{[v_1]_2, [v_2]_2, ..., [v_n]_2\}$
P_1 **Output**: sorted label vector L

1. From $i = 0$, P_1 and P_2 swap the i-th and i+1-th elements of $[V]_1, [V]_2$ and L if $SC(([v_i]_1, [v_{i+1}]_1), ([v_i]_2, [v_{i+1}]_2))$ returns 1.
2. P_1 outputs the sorted label vector L.

Protocol 5 Main Protocol

Server S **Input**: dataset $D = \{d_1, d_2, ..., d_n\}$ and label $L = \{l_1, l_2, ..., l_n\}$
Client C **Input**: query $Q = \{q_1, q_2, ..., q_p\}$
S **Output** to C: result $R = \{r_1, r_2, ..., r_p\}$

1. S converts $D \to [D]_1, [D]_2$; C converts $Q \to [Q]_1, [Q]_2$, where $\langle d_i \rangle_m$ is the m-th attribute of the i-th instance in the dataset D and $\langle q_j \rangle_m$ is the m-th attribute of the j-th instance in the query Q.
2. S sends $[D]_2$ to C and C sends $[Q]_1$ to S.
3. For each attributes, S and C obtains $([\triangle_{i,j,m}]_1, [\triangle_{i,j,m}]_2) \leftarrow PD(I_S, I_C)$ between each pair of instances where $I_S = ([\langle d_i \rangle_m]_1, [\langle q_j \rangle_m]_1)$ and $I_C = ([\langle d_i \rangle_m]_2, [\langle q_j \rangle_m]_2)$.
4. S computes $[dist_{i,j}]_1 = \sum_{h=1}^{m} [\triangle_{i,j,h}]_1$ and C does $[dist_{i,j}]_2 = \sum_{h=1}^{m} [\triangle_{i,j,h}]_2$.
5. For each query instances, S gets the sorted label vector $L'_j \leftarrow SS(([dist_{i,j}]_1, L), [dist_{i,j}]_2)$.
6. S set the major label of the k-th nearest data in each L'_j to r_j and outputs $R = \{r_1, r_2, ..., r_p\}$ to C.

V. Data Preprocessing for kNN by Secret Sharing

In order to reduce more calculation burden, hoping that our construction can save more computing time. In this section, we would like to join some data preprocessing methods to refine the naive scheme. First of all, we will partition the dataset, then bring a data structure into this scheme for storing data. In this way, the amount of data can be reduced, and more calculation time can be saved while maintaining a certain degree of accuracy. In the following, some techniques are formally introduced, and later we will use these techniques in our algorithms.

1) Partition: There are many ways to partition a dataset. The first step of partitioning is to find the center point. So, we recommend two ways below. One is to randomly select one data from the dataset. Another is to sample elements over distribution and find one point in a more distributed interval. Then, we will set this point as the center point and calculate the data within the distance of a radius value to divide the block.

2) Datum Point: In the tree preprocessing algorithm, we will need to set a datum point to calculate the distance between the data and the datum point, and then construct a tree structure based on the calculated distance. The method of setting the datum point is to randomly determine a point in the dataset.

3) Information Gain (IG): In the decision tree preprocessing algorithm, we will need to give an Information Gain for measuring the ability of a given attribute distinguishing the training instances. If the Information Gain is large, then this attribute is very important for the classification, and the data can be classified more cleaner.

Let A denote an attribute used by the node for separating out m child nodes, D_p, D_j is the data of the parent node and the j-th child node, and N_p, N_j represent the data quantity of them. With an impurity measure I stands for the degree of completeness of the classification, we can present the Information Gain for an attribute A as follows. $IG(D_p, A) = I(D_p) - \sum_{j=1}^{m} (\frac{|N_j|}{|N_p|} * I(D_p))$

A. Attempt: Tree-based Method

First of all, we partition the dataset and obtain the representatives. Then, we calculate the distances between the representative and the datum point and build a tree based on the distance. The details of our tree preprocessing algorithm are shown in Algorithm 1.

After the tree is produced, we can easily get the nearest partition block from a leaf node that has a subset of the dataset. Finally, we can perform the main protocol (Protocol 5) on the selected block and get the classification result. The schematic is shown as Figure 1. Unfortunately, we found that the tree-based method with the reduction of the dataset has accuracy problems, so we changed to constructing a decision tree as the solution.

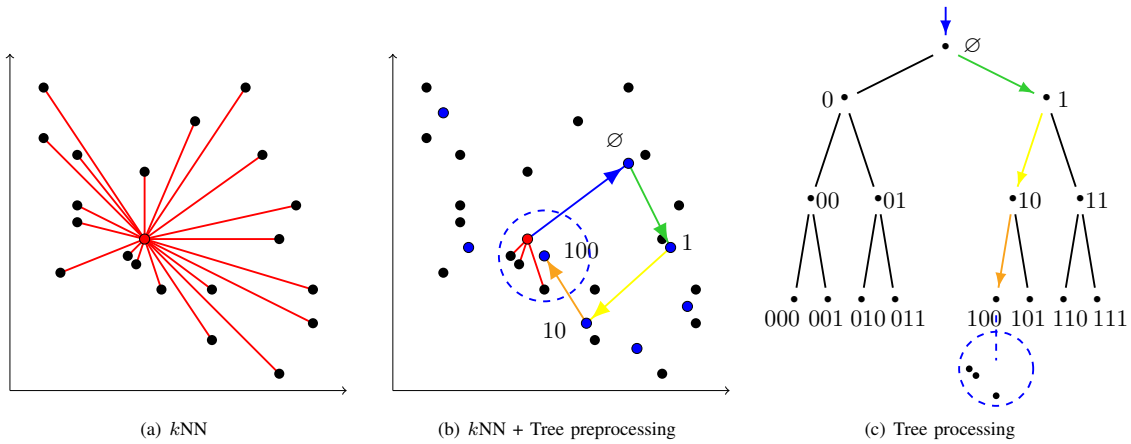
Algorithm 1 Tree Preprocessing Algorithm (TP)

Input: dataset
Output: total level of the tree j

1: Divide the dataset into several blocks $block_i^j$ via partition, where j is the number of levels of the tree; i is the number of nodes.
2: **while** number of $block^j > 1$ **do**
3: Define representative through the average value of all data in each block: $Average(block_i^j) \to rep_i^j$.
4: Calculate the distance d_i^j between the datum point and the representative.
5: Sort the blocks through the calculated distance d_i^j to form a node $node_i^j$: $Sort(\{rep_i^j\}_{i=1}^k) \to \{node_i^j\}_{i=1}^k$.
6: Merge two nodes into a block to form the upper node: for each i $Define\{node_{2i-1}^j, node_{2i}^j\} \to block_i^{j+1}$
7: **end while**

B. Decision Tree-based Method

The decision tree can select the attributes with the best classification ability as the internal nodes of the tree, and



(a) kNN (b) kNN + Tree preprocessing (c) Tree processing

Figure 1: Schematic diagram

generate corresponding branches from all the different data of the internal nodes. This method allows us to make better predictions of partitions and improve accuracy.

Similarly, we first partition the dataset and obtain the representatives. Then we use these representatives to build a decision tree and record the leaf node corresponding to each representative. The details of our decision tree preprocessing algorithm are shown in Algorithm 2.

After the decision tree is produced, we convert the decision tree into the form of secret sharing. So we can easily get the predicted leaf node for the query data by protocol 2, then find the nearest representative in the leaf record. Note that each leaf may have multiple representatives. Finally, we perform the PPkNN with these collected instances and obtain the classification result.

Algorithm 2 Decision Tree Preprocessing Algorithm (DTP)

Input: dataset and attribute list
Output: decision tree and leaf record

1: Divide the dataset into several blocks $block_i$ via partition, where i is the number of blocks.
2: Define representative through the average value of all data in each block: $Average(block_i) \rightarrow rep_i$.
3: Collect all representatives as a new dataset D_0.
4: Create a root node with the representative dataset D_0.
5: **while** number of $attributeList > 1$ **do**
6: Calculate the Information Gain of each attribute with current node dataset: $IG(D, m_j) \rightarrow gain_j$, where j is the number of an attribute in the attribute list.
7: Get the attribute m with the largest information gain, and split the current node dataset with this attribute m.
8: Remove the attribute m from $attributeList$: $attributeList - m \rightarrow attributeList$.
9: **end while**
10: Record the leaf node correspond to the representatives and data : for each leaf node N_k with dataset D_k , $Record + (k, (rep_i, block_i)) \rightarrow Record$, if $rep_i \in D_k$.

C. Security Analysis

Lemma 1. *Secure comparison protocol is secure.*

The security of the SC protocol can be briefly introduced by RG protocol and Protocol 1. In this protocol, the relationship of the values has been implied with the parameter l, and secret sharing provides a safe way to calculate without knowing the secret, which makes us obtain the comparison without any privacy leaking. Thus, we can claim that the secure comparison protocol is secure.

Theorem 1. *Naive Construction and Decision Tree-Based are secure.*

Proof. For the Naive Construction, it is consists of a large number of comparisons in Protocol 3 and Protocol 4. Based on Lemma 1, we can know that the calculation in the main protocol (Protocol 5) is secure. So we can consider an adversary $\mathcal{A}$ who can eavesdrop the communication or even colluding with the server S or the client C to get the dataset D or query data x. Let π be our scheme and function f represent the privacy leakages. We can get the probabilities below.

- $Pr[\mathcal{A}_S^{\pi}(D) \rightarrow f(x)] < \mathsf{negl}$
- $Pr[\mathcal{A}_C^{\pi}(x) \rightarrow f(D)] < \mathsf{negl}$

For the decision tree-based method, the decision tree preprocessing algorithm (Algorithm 2) is noninteractive, which may be secure in preprocessing. Despite the query data maintain privacy preserve, the decision tree occurs a little inevitable leakage in the sharing decision node. In other words, $\mathcal{A}$ can not know the data value, but $\mathcal{A}$ can observe the attribute of the data while using the decision tree to choose the nearest representative.

VI. Discussions

A. Theoretical Analysis

In the naive construction, we will take $O(m \cdot n)$ to calculate the distance between the m attributes query data and all n instances in the dataset. Then, take $O(n^2)$ for the PPkNN

classifier to compare the distance shares and find the k nearest data. With the decision tree-based method, we can reduce the amount of data by predicting the partitions of the dataset on the decision tree consuming $O(\log m)$, where m is the number of the attibutes. Then, only need $O(m \cdot p)$ comparison for the PPkNN classifier, where p is the number of instances in the nearest representative. A schematic diagram is shown in Figure 1.

B. Experimental Results

We leverage a couple of datasets from the UCI repository to prove the applicability on real datasets and test our scheme over these datasets with different parameters of partitions shown in Table I. At least 10 epochs of random test data are selected and evaluated with each dataset. The average performance of PPkNN and the decision tree-based (tree generation and PPkNN classification) is shown in Table II.

Figure 2 shows the accuracy comparison between the traditional kNN and the decision tree-based method on different datasets with the parameter setting as Table I. Our scheme holds a certain degree of accuracy by inheriting the classification way of a decision tree to find the nearest representative. To put it simply, it finds the nearest representative in the decision tree leaf node and applies kNN to find the nearest neighbor on them.

Table I: Parameters of partition for different datasets.

Dataset	Total instances	Num. of attributes	Num. of representatives	Avg. instance in representatives
Iris	150	4	50	19.7
Wine	178	13	100	14.28
Breast cancer	569	30	300	20.05
Digits	1797	64	500	90.15
Mushroom	8124	22	1000	39.48
Nursery	12960	8	1250	165.42

Table II: Time consumption in different datasets.

Dataset	Without Preprocessing PPkNN time	Decision Tree-base Method tree gen. time	Decision Tree-base Method PPkNN time
Iris	1.92 (s)	0.33 (s)	0.64 (s)
Wine	8.75 (s)	0.95(s)	2.62 (s)
Breast cancer	199.76 (s)	8.41 (s)	26.87 (s)
Digits	4209.67 (s)	222.58 (s)	397.90 (s)
Mushroom	24121.25 (s)	180.03 (s)	848.61 (s)
Nursery	22911.80 (s)	447.44 (s)	810.79 (s)

VII. Conclusions

In this paper, we presented an efficient PPkNN on secret sharing, which holds a certain degree of accuracy and enabled us to run faster in the PPkNN algorithm by using a portion of the dataset from the tree structure. Finally, experiment results showed that our scheme maintains data privacy in real datasets and has higher efficiency than the naive scheme.

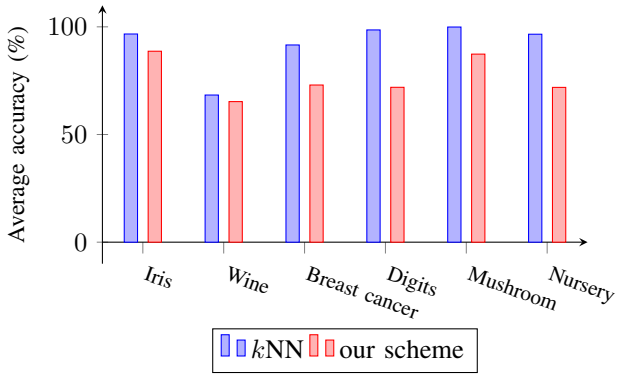


Figure 2: Average accuracy in different dataset

Acknowledgements

This work was supported in part by Ministry of Science and Technology of Taiwan (Nos. 106-2218-E-115-008-MY3, 109-2628-E-155-001-MY3 and 110-2218-E-004-001-MBK).

References

[1] H. W. Lim, G. S. Poh, J. Xu, and V. Chittawar, "PrivateLink : Privacy-preserving integration and sharing of datasets," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 564–577, 2019.

[2] M. Mahmoud, K. Rabieh, A. Sherif, E. Oriero, M. Ismail, E. Serpedin, and K. Qaraqe, "Privacy-preserving fine-grained data retrieval schemes for mobile social networks," *IEEE Transactions on Dependable and Secure Computing*, vol. 16, no. 5, pp. 871–884, 2017.

[3] T. T. Nguyên, X. Xiao, Y. Yang, S. C. Hui, H. Shin, and J. Shin, "Collecting and analyzing data from smart device users with local differential privacy," *arXiv preprint arXiv:1606.05053*, 2016.

[4] N. Wang, X. Xiao, Y. Yang, J. Zhao, S. C. Hui, H. Shin, J. Shin, and G. Yu, "Collecting and analyzing multidimensional data with local differential privacy," in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 2019, pp. 638–649.

[5] A. Moldagulova and R. B. Sulaiman, "Using knn algorithm for classification of textual documents," in *2017 8th International Conference on Information Technology (ICIT)*, 2017, pp. 665–671.

[6] W. Xing and Y. Bei, "Medical health big data classification based on knn classification algorithm," *IEEE Access*, vol. 8, pp. 28 808–28 819, 2020.

[7] P. Mohassel and Y. Zhang, "Secureml: A system for scalable privacy-preserving machine learning," in *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 19–38.

[8] M. Shen, B. Ma, L. Zhu, X. Du, and K. Xu, "Secure phrase search for intelligent processing of encrypted data in cloud-based iot," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 1998–2008, 2018.

[9] H. Li, D. Liu, Y. Dai, T. H. Luan, and S. Yu, "Personalized search over encrypted data with efficient and secure updates in mobile clouds," *IEEE Transactions on Emerging Topics in Computing*, vol. 6, no. 1, pp. 97–109, 2015.

[10] M. Kantarcıoğlu and C. Clifton, "Privately computing a distributed k-nn classifier," in *Knowledge Discovery in Databases: PKDD 2004*. Springer Berlin Heidelberg, 2004, pp. 279–290.

[11] H. Rong, H.-M. Wang, J. Liu, and M. Xian, "Privacy-preserving k-nearest neighbor computation in multiple cloud environments," *IEEE Access*, vol. 4, pp. 9589–9603, 2016.

[12] J. Park and D. H. Lee, "Privacy preserving k-nearest neighbor for medical diagnosis in e-health cloud," *Journal of Healthcare Engineering*, vol. 2018, 2018.

[13] M. Sun and R. Yang, "An efficient secure k nearest neighbor classification protocol with high-dimensional features," *International Journal of Intelligent Systems*, vol. 35, no. 11, pp. 1791–1813, 2020.

[14] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.