Protocols and Data Structures for Knowledge Discovery on Distributed Private Databases

by

Amirbekyan Artak
M.Sc(Mathematical Modelling and Scientific Computing)(TU Kaiserslautern),
Diploma of Mathematics(Yerevan State University)

A thesis submitted in fulfillment of the requirements
for the degree of Doctor of Philosophy

School of Information and Communication Technology
Griffith University

July 2007

Certificate of Originality

I hereby certify that the work embodied in this thesis is the result of original research and has not been submitted for a higher degree at any other University or Institution.

(Signed) ______________________

Amirbekyan Artak

Approval

Name:	Amirbekyan Artak
Degree:	Doctor of Philosophy
Thesis Title:	Protocols and Data Structures for Knowledge Discovery on Distributed Private Databases
Submission Date:	2007
Supervisor:	Professor Vladimir Estivill-Castro
Co-supervisor:	Professor Rodney Topor
External examiners:	First Second

Acknowledgements

I would like to sincerely thank my supervisor, Professor Vladimir Estivill-Castro. This thesis would not have been possible without his tireless assistance and support.

I am very thankful to fellow PhD students, particularly Joel Fenwick for his regular assistance, valuable contributions and just for being a friend.

This research was supported in part by ARC (Australian Research Council) Discovery Research Grant. Project D P0451129: “Privacy Preserving Data Mining as Autonomous Data Analysis Expands into Safeguarding from Threats like Crime and Terrorism”.

Finally, I thank my wife for her encouragement and support, my two boys for helping me forget about work at home. Also my parents, all my relatives and friends whose help was always there when it was needed.

To my wife and my parents.

List of Outcomes Arising from this Thesis

International Journals with full paper refereed

1. (waiting for notification) A. Amirbekyan and V. Estivill-Castro. *Practical protocol for Yao's millionaires problem enables secure multi-party computation of metrics and efficient privacy-preserving k-NN for large data sets.* Knowledge and Information Systems. An International journal.

International conferences and workshops with full paper refereed

1. A. Amirbekyan and V. Estivill-Castro. Privacy preserving *DBSCAN* for vertically partitioned data. In *IEEE International Conference on Intelligence and Security Informatics, ISI 2006*, pages 141-153, San Diego, CA, USA, May 23-24 2006. Springer Verlag Lecture Notes in Computer Science 3975.

2. A. Amirbekyan and V. Estivill-Castro. The privacy of k-NN retrieval for horizontal partitioned data – new methods and applications. In J. Bailey and A. Fekete, editors, *Eighteenth Australasian Database Conference (ADC2007)*,pages 33-42, Conferences in Research and Practice in Information Technology (CRPIT), Ballarat, Victoria, Australia, January 2007. CORE, Australasian Computer Society.

3. A. Amirbekyan and V. Estivill-Castro. Privacy-Preserving k-NN for Small and Large Data Sets. *IEEE International Conference on Data Mining, ICDM 2007, Workshop on PADM*, Omaha, NE, USA.

4. A. Amirbekyan and V. Estivill-Castro. Privacy-Preserving Regression Algoritms. *The 3rd WSEAS International Symposium on Data Mining and Intelligent Information Processing, ISDM 2007,* Beijing, China.

5. A. Amirbekyan and V. Estivill-Castro. A New Efficient Privacy-Preserving Scalar Product Protocol. *The Australasian Data Mining Conference, AusDM 2007.* Gold Coast, Australia.

Abstract

Data mining has developed many techniques for automatic analysis of today's rapidly collected data. Yahoo collects 12 TB daily of query logs and this is a quarter of what Google collects. For many important problems, the data is actually collected in distributed format by different institutions and organisations, and it can relate to businesses and individuals. The accuracy of knowledge that data mining brings for decision making depends on considering the collective datasets that describe a phenomenon. But privacy, confidentiality and trust emerge as major issues in the analysis of partitioned datasets among competitors, governments and other data holders that have conflicts of interest. Managing privacy is of the utmost importance in the emergent applications of data mining. For example, data mining has been identified as one of the most useful tools for the global collective fight on terror and crime [80].

Parties holding partitions of the database are very interested in the results, but may not trust the others with their data, or may be reluctant to release their data freely without some assurances regarding privacy. Data mining technology that reveals patterns in large databases could compromise the information that an individual or an organisation regards as private. The aim is to find the right balance between maximising analysis results (that are useful for each party) and keeping the inferences that disclose private information about organisation or individuals at a minimum.

We address two core data analysis tasks, namely clustering and regression. For these to be solvable in the privacy context, we focus on the protocol's efficiency and practicality. Because associative queries are central to clustering (and to many other data mining tasks), we provide protocols for privacy-preserving k-near neighbour (k-NN) queries. Our methods improve previous methods for k-NN queries in privacy-preserving data-mining (which are based on Fagin's A0 algorithm) because we do leak at least an order of magnitude less candidates and we achieve logarithmic performance on average. The foundations of our methods for k-NN queries are two pillars, firstly data structures and secondly, metrics. This thesis provides protocols for privacy-preserving computation of various common metrics and for construction of necessary data structures.

We present here new algorithms for secure-multiparty-computation of some basic operations (like a new solution for Yao's comparison problem and new protocols to perform linear algebra, in particular the scalar product). These algorithms will be used for the construction of protocols for different metrics (we provide protocols for all Minkowski metrics, the cosine metrics and the chessboard metric) and for performing associative queries in the privacy context. In order to be efficient, our protocols for associative queries are supported by specific data

structures. Thus, we present the construction of privacy-preserving data structures like *R*-Trees [42, 7], *KD*-Trees [8, 53, 33] and the *SASH* [8, 60].

We demonstrate the use of all these tools, and we provide a new version of the well known clustering algorithm *DBSCAN* [42, 7]. This new version is now suitable for applications that demand privacy. Similarly, we apply our machinery and provide new multi-linear regression protocols that are now suitable for privacy applications.

Our algorithms are more efficient than earlier methods and protocols. In particular, the cost associated with ensuring privacy provides only a linear-cost overhead for most of the protocols presented here. That is, our methods are essentially as costly as concentrating all the data in one site, performing the data-mining task, and disregarding privacy. However, in some cases we make use of a third-trusted party. This is not a problem when more than two parties are involved, since there is always one party that can act as the third.

Notation

SMC — Secure Multiparty Computation.

PP — Privacy-Preserving.

DM — Data Mining.

OLTP — On-line Transaction Processing.

SASH — Spatial Approximation Sample Hierarchy.

DNSP — Distributed non-private setting.

$A \times B$ — The Cartesian product of the sets A and B; that is, the set of all ordered pairs (a, b) with $a \in A$ and $b \in B$.

$\Re$ — The set of real numbers.

$\vec{v}$ — A vector v.

$\vec{a}^T \cdot \vec{b}$ — Scalar product.

Databases used in experiments[1]:

Database 1 — The CoIL 2000 Challenge [97] dataset contains information on customers of an insurance company. The data consists of 86 variables with 5821 records and includes product usage data and socio-demographic data derived from zip area codes.

Database 2 — The Census-Income Database holding multivariate PUMS census data from the Los Angeles and Long Beach areas for the years 1970, 1980, and 1990 (from KDD UCI repository). Combining test and training databases we get 299,285 records with 40 dimensions/attributes.

Database 3 — The data set named *Histogram* corresponds to a colour histogram. The histogram data set has dimension 64 and 12,103 records.

Database 4 — The data set *Stock* corresponds to a stock market price. *Stock* has dimension 360 and 6,500 records coming from to different companies.

Database 5 — An aerial image dataset with dimension 60 and 275,465 records.

[1]Database 1 and Database 2 are in the KDD UCI repository. We thank Professor B.S. Manjunath (University of California, CA, USA) for donating Database 3, Database 4 and Database 5.

Glossary of Terms

Commodity server — The commodity server is a third party. Alice and Bob can send requests to the commodity server and receive data (called commodities) from the server, but the commodities should be independent of Alice's or Bob's private data. The purpose of the commodities is to help Alice and Bob conduct the desired computation.

Semi-trusted third party — A third party is semi-trusted if:

1. It is not possible (for this third party) to derive any private information from other parties.
2. The third party does not collude with any of the parties.
3. The third party follows the protocol correctly.

Homomorphic encryption — Let $E()$ be an encryption scheme. That is, if x is plain, then $E(x)$ is encrypted. The encryption scheme E is called additive homomorphic if it has the following property:

$$E(x_1) \times E(x_2) = E(x_1 + x_2).$$

Paillier [83] encryption scheme is an example of additive homomorphic encryption. This can be generalised as well

$$E(x_1) \times E(x_2) \times \cdots \times E(x_n) = E(x_1 + x_2 + \cdots + x_n).$$

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Why Data Mining ?

In the past two decades, the amount of information or digitaly stored data have dramatically increased. The accumulation of data has taken place at an explosive rate. In companies like Yahoo, Google and eBay, the amount of data is quite stunning. Previously Terabytes and Gigabyte sizes were in usage. Now we have gone up to Petabytes and Exabytes. In 2006, Google reached 2 Petabytes of disk space [34]. The first commercially-available Petabyte Storage Array was launched by the EMC corporation in January 2006, with an approximate cost of USD $4 million [84]. We now even find an expression claiming that "all words ever spoken by human beings" could be represented in text data by approximately 5 Exabytes of data [82, 73]. In 2003, University of California, Berkeley, reported that in 2002 alone, "telephone calls worldwide on both landlines and mobile phones contained 17.3 Exabytes of new information if stored in digital form", and "it would take 9.25 Exabytes of storage to hold all U.S. [telephone] calls each year [82]".

Electronic data collecting devices, such as point-of-sale or remote sensing devices, have also contributed to this explosion of available data.

Data is collected - what is next? The problem is what to do with this valuable resource? It was recognised that information stored is a core component of business decisions and the accumulated data can widen the possibilities for informed decision making.

Database management systems provided efficient access to the data and traditional on-line transaction processing systems, OLTPs, are good at putting data into databases quickly, safely and efficiently. However, they are not not designed for delivering useful analysis in return. Analysing data can provide further knowledge about a business by going beyond the data explicitly stored or queried. This is where Data Mining (DM) or Knowledge Discovery in Databases (KDD) has obvious benefits for any private company or government organisation.

The term Data Mining has been applied to many forms of data analysis. Some of the numerous definitions of Data Mining, or Knowledge Discovery in Databases are:

> "Data Mining, or Knowledge Discovery in Databases (KDD) as it is also known, is the nontrivial extraction of implicit, previously unknown, and potentially useful information from data. This encompasses a number of different technical approaches, such as clustering, data summarisation, learning classification rules, finding dependency networks, analysing changes, and detecting anomalies" [52].

> "Data mining provides insight into hidden patterns in your organizations data and enables you to develop predictive models that help you anticipate change, so you can make informed decisions and take action" [2].

Frankly, data mining is concerned with the analysis of data and the use of various techniques for finding patterns, meaningful information and regularities in data sets. It is the software which is responsible for finding the patterns by identifying the underlying rules and features in the data. The idea is that it is possible to find gold in unexpected places as the data mining software extracts patterns not previously discernible. A SQL query is usually used to retrieve some specific data while data miners might not even be exactly sure of what information/pattern they are going to get.

1.2 Data Mining Techniques

Data mining techniques may be classified by the purpose they are used for. With the wide range of data mining applications, the most used techniques include classificaion, association rules mining, sequential patterns discovery and clustering.

1.2.1 Classification

Classification is a form of data analysis that can be used to extract models describing data classes. The owner of the data defines classes depending on some attributes of the data, then the job for classification is to automatically predict the class of the new tuple. For instance, a bank can classify clients as risky or safe for loan purposes, depending on the attributes that describe every client. With this tools, the bank obtains a model for classification. Then new loan applicants can be classified automatically by this model.

The classical classification methods include decision trees, Bayesian classification, k-Nearest-Neighbour classifiers. Classification from a machine learning perspective is described in several books [99, 57].

1.2.2 Associations

Given a set of records, that contains some number of items from a given collection of items, association rule mining is used to extract some patterns that exist between items in the set of records. These patterns can be expressed by rules like:
"60% of all the records that contain items A, B and C also contain items D and E". The specific percentage of occurrences (in this case 60) is called the confidence factor of the rule.

Association rule mining was first proposed by Agrawal, Imielinski, and Swami [5].

1.2.3 Sequential/Temporal Patterns

Data Mining tools can be applied for identifying, for instance, trends in the set of records over a period of time. With this type of tool we would be able to identify the purchases repeatedly made by some customers over a period of time. Obviously, this information can be used for mail advertisement. Sequential patterns analysis can also discover sets of items that usually follow, for instance, microwave oven purchases.

1.2.4 Clustering/Segmentation

Clustering and segmentation are the tools for partitioning the data in such a way, that all the members of each group are close to each other according to some metric. A cluster is a set of objects in the data grouped together by some similarity measure.

Clustering of objects by similarity is a very powerful technique. It translates some intuitive grouping into a quantitative grouping.

When clustering is automated, then the system has to discover its own classes i.e. the identify clusters in the database.

There are a number of approaches for forming clusters. One approach is to form rules which dictate membership in the same group based on the level of similarity between members. Another approach is to build set functions that measure some property of partitions as functions of some parameter of the partition.

Well known methods for clustering include K-means, K-medoids, $DBSCAN$, $CURE$ and $OPTICS$. Various surveys and books also offer descriptions of more clustering methods [18, 58, 72].

1.3 Data Structures, Similarity Search and the Importance of Associative Queries and Metrics

In computer science, data structures are an important way of organising data in a computer so that it can be used efficiently. There are many different data structures used to organise data in computers. Some data structures are similar to tree diagrams because they are good for representing relationships between data. Other structures are good for ordering data in a particular way like a list of employees. Each data structure has unique properties that make it well suited to give a certain view of the data or for some particular job that it was designed for. Here we will present some classical data structures like *KD*-Trees and *R*-Tree as well as a recent, but successful data structure the *SASH*.

For most data mining algorithms, the data is encoded as vectors in high dimensional space[1] For these algorithms, a measure of similarity (or dissimilarity) is necessary, and many times fundamental for their operation. Similarity queries on multi-dimensional data are usually implemented by finding the closest attribute-vector(s) to the attribute-vector of the query data. In such settings, information retrieval under the vector model must typically be implemented as k-nearest-neighbour (k-NN) queries, whose result consists of the k items closest to the query vector according to the similarity measure. This type of query is known as a nearest neighbour (NN) query [85] and it has been studied in the past [11, 19, 26]. Another closely related query is the ϵ-range query that returns all vectors that are within the ϵ-neighbourhood of the query vector $\vec{q}$.

Similarity search is widely used as a common form of query in modern database applications such as multimedia information systems [90], geographical information systems (GIS) [25], time-series databases [49], medical imaging [74], and bioinformatics [65]. The similarity between two objects is defined with a distance function, e.g., Euclidean distance, between the corresponding attribute-vectors. For example, in image databases, the similarity search can be used for retrieving the most similar images to a given image [10]. 3D shape histograms are used in

[1] Attribute-vectors are the common input for learning algorithms like decision trees, artificial neural network or for clustering algorithms like K-Means or *DBSCAN*.

molecular biology to find similar 3D proteins [9]. Consider a database consisting of DNA sequences of people with some chronic disease. Users can query this database to check whether there are similarities between their DNA sequences and the ones in the database.

While range queries enable distance-based clustering (as in *DBSCAN* with R-Trees) and outlier detection [94], k-NN queries also enable Local Outlier Detection [87], Shared Nearest Neighbour Clustering [87] and k-NN Classification [8, 67, 87].

When the dataset is large, data structures that efficiently support k-NN search are essential to many applications. This family includes classic search structures like KD-Trees [16] and R-Trees [56], and newer data structures and techniques such as SR-Trees [71], X-Trees [17] and iDistance [103]. The central role of these indices in data mining algorithms is illustrated by the role that R-Trees play in the efficiency of the popular data mining clustering algorithm *DBSCAN* [42, 7]. The original *DBSCAN* uses R-Trees for performing clustering tasks. The R-Trees are used for applying ϵ range queries and depending on the number of vectors found, some vectors are considered as core vectors. The clusters in a sense represent accumulation of the core vectors. Thus, if a data structure such as R-Trees were not used to support efficient ϵ range queries, then naturally the *DBSCAN* algorithm would be very slow.

1.4 Why Privacy Preserving Data Mining ?

Why do we carry out a census? Mainly, for government and other public and private organisations to make sensible decisions. Data Mining (DM) is about making sensible decision after analysing large volumes of data. The sociological and legal context of Data Mining's threats to privacy and the formulation of information privacy policies or security policies are not our focus. We address the challenges of exploratory Data Mining tools as they may correlate, make inferences and disclose confidential, sensitive facts about individuals. For instance, a central task in Data Mining is inductive learning; this takes as input a training data set and produces as output a model (classifier) which is then applied to unseen cases to predict some important, and perhaps confidential attribute (for example, customer buying power or medical diagnosis).

Data mining technology allows the analysis of large amounts of data. Analyses of personal data, or analyses of corporate data (by competitors, for example) create threats to privacy [44].

Moreover, never have globalization and international collaborations placed as much demand on partnerships between governments and/or corporations as they do today. Data mining has been identified as one of the most useful tools for the fight on terror and crime [80]. However, the information needed resides with many different data holders that must share their data with each other; thus, data privacy becomes extremely important. Parties may not trust each other, but all parties are aware of the benefit brought by such collaboration. In the privacy preserving model, all parties of the partnership promise to provide their private data to the collaboration, but none of them wants the others or any third party to learn much about their private data.

Privacy-preserving clustering, for instance, is more apparent in settings where different data holders have different data about the same individuals (vertically partitioned data [92]). For example, a government agency may have classified some individuals as law-obeying citizens, some as potentially dangerous individuals, and others as certainly dangerous individuals based on attributes available to the law enforcing agencies. However, a more accurate classification (clustering) could be obtained if data about the financial transactions of individuals was available as well. Then, police resources could be more focused for more promising (and perhaps preventive) investigations. But financial transactions or phone records may be the ownership of banks or phone companies that may or may not be obliged to disclose them (for example, in some countries these records are not to be made available unless the individual is actually being charged).

1.4.1 Significance

Nowadays, computers can manipulate large databases and perform many data analysis tasks using data-mining techniques. Our purpose is, during such autonomous data analysis, to preserve privacy of individuals and corporations. Data is now available from companies, shops, medical clinics, hospitals and it can be used to detect patterns to identify individuals who can be dangerous to society or obtain information which will help to make preventative decisions. But such a task is not possible without some level of privacy protection as the companies or hospitals policies protect private information. Thus a method that can achieve a balance on knowledge discovery and privacy protection is highly desirable.

Analysis of personal data or corporate data (by competitors, for example) creates threats to information privacy and may allow for easier data surveillance. With the current emphasis on intelligence for safeguarding modern societies from crime and terrorism, Data Mining technology is to be applied to analyse large amounts of digitally recorded data about the activities and operations of individuals and organisations for spotting out the potentially dangerous [80] ones. This threatens privacy and the values of democratic societies. On the other hand in the US "Both the Federal Bureau of Investigation and the new Department of Homeland Security have identified data mining as a key component in combating crime and terrorism in the 21st century" [98]. In the UK, "We have been very successful in using DM to analyse patterns in crime"and "Being able to mine text opens up new possibilities for us and significantly expands our capabilities for crime pattern analysis" [3] (Inspector R. Adderley of West Midland's Police Department UK).

Can we carry out research based on facts ? Researchers feel that privacy regulations enforce inconsistent restrictions on data exploration, and, in some cases, ruin the data. How could planning decisions be taken if census data was not collected? How could epidemics be understood if medical records were not analysed? Individuals benefit from data collection efforts via the process of building knowledge that guides society. Privacy protection cannot be achieved simply by restricting data collection or restricting the use of information technology.

1.4.2 Privacy Preserving Clustering as an Example

Several core tasks of knowledge discovery in databases have been defined in the literature [79], that includes class identification; that is, the grouping of the objects of a database into meaningful subclasses. Clustering algorithms are attractive for the task of class identification, discovery and formation. However, this application to large databases raises the following requirements for clustering (see Section 1.2.4) algorithms:

1. Little or no human intervention or domain knowledge when supplying input parameters to allow the discovery process to be exploratory and free of human bias.
2. Flexible modelling to allow for arbitrary shape of clusters.
3. Efficiency on large databases.

The task here is not only to have clustering algorithms satisfying the above requirements, but to enable clustering algorithms to ensure a level of privacy.

However, very few clustering methods have been devised for the privacy-preserving (PP) context. Recently, such clustering algorithms as K-means [93], K-medoids [43] and *DBSCAN* [7] have been presented in the privacy context.

Clustering algorithms are always supported with the appropriate data structures for efficiency, so privacy-preserving clustering without support of privacy-preserving data structures would not be practical. However, only a few of the popular data structure like *KD*-Trees [16, 8], *R*-Trees [56, 7] and *SASH* [60, 59, 8] have privacy-preserving versions.

Traditional data mining algorithms for clustering usually depend on associative queries, which use different types of data structures and metrics to enhance efficiency (see Figure 1.1).

Hence, it is natural to expect that the privacy-preserving data mining clustering algorithms would be dependant on privacy-preserving associative queries, privacy-preserving data structures and privacy-preserving metrics (see Figure 1.2).

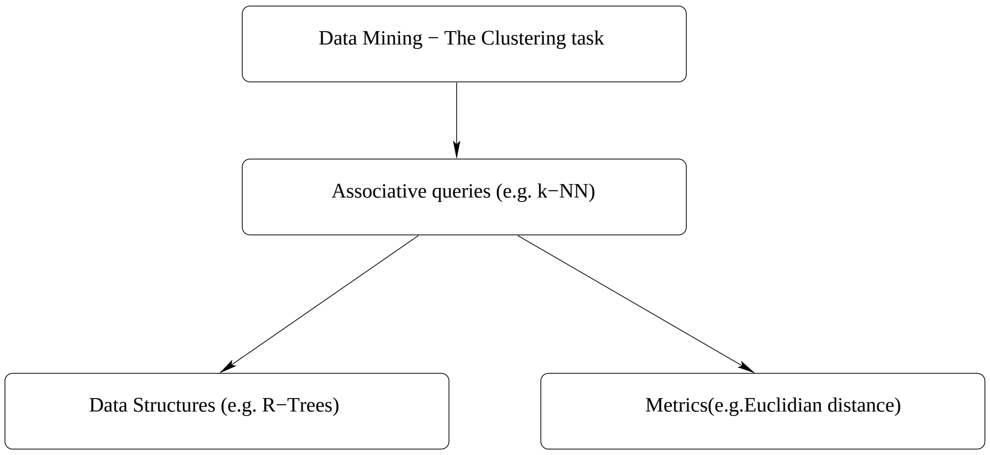

Data Mining − The Clustering task
Associative queries (e.g. k−NN)
Data Structures (e.g. R−Trees)
Metrics(e.g.Euclidian distance)

Figure 1.1: Traditional data-mining clustering task.

1.4.3 Aims and Contributions

In this work we focus on three aspects.

- Popular data mining algorithms such as clustering and regression analysis, but we bring them to the privacy context.
- This adaptation will require changes to the algorithms and the data structures that support them, so that associative queries can operate preserving privacy.
- Developing new operations and methods in the secure multiparty computational area. This thesis deals with two well known cases of data partitioning: horizontal and vertical partitioned data.

The issue of data partitioning is closely linked to the issue of privacy. If there were no privacy, then every party would have access to the data and effectively no partition would exists.

If parties have obtained their data separately and have no privacy concerns, they could merge the data and the partition would dissolve.

Thus, one can not divorce a debate on privacy from some debate about the spread of data and/or knowledge between the parties.

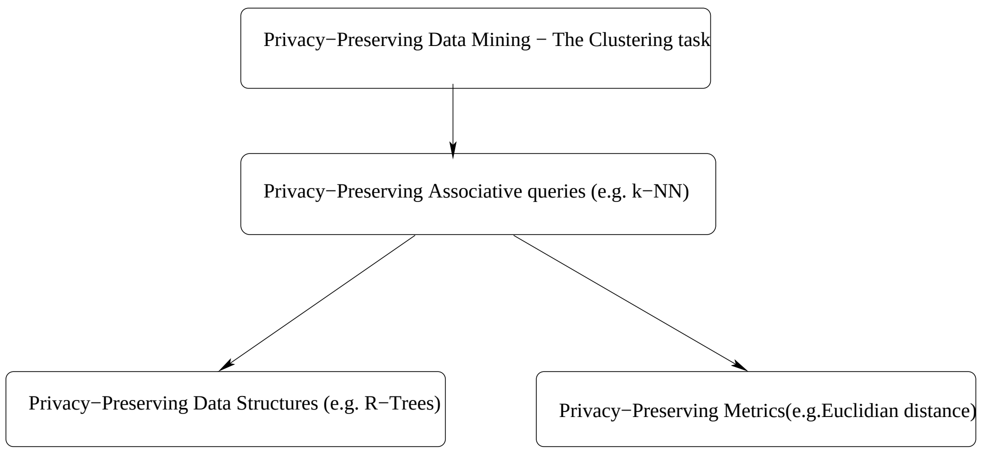

Privacy−Preserving Data Mining − The Clustering task
Privacy−Preserving Associative queries (e.g. k−NN)
Privacy−Preserving Data Structures (e.g. R−Trees)
Privacy−Preserving Metrics(e.g.Euclidian distance)

Figure 1.2: Privacy-preserving data mining clustering task

As a contribution to the field of privacy-preserving data mining this thesis will

- extend and migrate existing clustering algorithms to the privacy preserving data mining context,
- develop more general tools in privacy-preserving data mining, in particular, rather than focus on each individual clustering algorithm, we revise the general data structures used in many algorithms, and
- develop general secure multiparty computational tools that will help algorithms that are not designed for privacy-preserving data mining to systematically be transformed to the its context.

There are two avenues in the privacy-preserving data mining:

- **Data Perturbation:** Perturb the data so that statistics are globally accurate but there is uncertainty about values for individuals.

  Data perturbation is one of the approaches for privacy protection in data mining. The main idea is to perturb the input before the mining [6], but at the same time, preserve the statistical nature of data while protecting the privacy of the data. However, this approach lacks a formal framework for proving how much privacy is guaranteed. Despite the existence of several models [6, 35, 47] for studying the privacy gain through perturbation, there is no formal way to model and quantify the privacy threat from mining perturbed data. Moreover, recently some evidence has been shown that for some data, and some kinds of noise, perturbation does not provide privacy at all [62, 68].

- **Secure Multi-Party Computation:** Data is distributed among parties, who compute a data mining task. They are all interested in using the union of the data but they learn nothing about each other's data.

  This approach for privacy preserving data mining uses cryptographic techniques, most often secure multi-party computation techniques [37, 55, 92]. This approach became very popular because cryptography offers a well dened model for privacy, which includes methodologies for proving and quantifying it. Moreover, there exist many tools of cryptographic algorithms that can be used for implementing privacy-preserving data mining algorithms. While secure multi-party computation (SMC) has an advantage over perturbation in that it provides accurate results and not approximation, it is a much slower method and requires considerable computation and communication overhead.

This thesis will focus only on the second avenue. That is, we will consider that the data is distributed among several parties, where parties are interested in performing data analysis tasks on the union of the data. Thus, Chapter 2 provides some building blocks for SMC. Data perturbation will no longer be the subject of discussion.

1.5 Balance Between Privacy and Practicality

Privacy is not without cost. One should always keep in mind, that in situations where privacy preservation is a concern, the algorithms that do not ensure some level of privacy will not be considered at all.

Privacy represents an additional constraint in the model of computation, and even theoretically, one cannot expect that the more restricted family of algorithms that ensure privacy will contain those algorithms that have extremely efficient complexity, but disregard privacy. Therefore, the additional cost of preserving privacy may not be trivial. Naturally, one may be willing to leak some information to reduce some cost. There will be situations (like a very small dataset), where any release of information could be considered very damaging. In such situations, the additional cost of privacy should be clearly an absolute priority. In a sense, these are relatively few values and releasing information is a large relative loss. But, if the dataset is very large, as is usually the case in data mining applications, we may accept that the cost of information leak (relative to the dataset size) and the necessity for efficiency justifies a small leakage of information as long as that leakage can be identified as inconsequential and innocuous.

The SMC literature has a general solution for all polynomially bound computations [55]. This generic "shares" solution computes $f(\vec{x}, \vec{y})$ for a polynomial-time f using private input $\vec{x}$ from Alice and private input $\vec{y}$ from Bob. Alice learns nothing about $\vec{y}$ except what can be computed from $f(\vec{x}, \vec{y})$ and similarly Bob learns nothing about $\vec{x}$ except what can be inferred from $\vec{y}$ and $f(\vec{x}, \vec{y})$. Why, if such a solution exists, is there so much interest in protocols for SMC? The first aspect is that the general solution requires f to be explicitly represented as a Boolean circuit of polynomial size. Even if represented as a circuit of polynomial size in its input, the input must be very small for the circuit to have practical polynomial size. This means, the sub-task that uses this result must be on small inputs, a constraint difficult to meet in data mining applications. Third, the constants involved are not small, so once the circuit is described the parties enter into a protocol, holding shares of the inputs to gates and shares of the outputs of gates. Fourth, the privacy-preserving literature shows the need for practical and efficient solutions that are not based on this general theoretical solutions [55]. It also shows that much more efficient solutions exist for special cases of f, for instance, solution for Yao's millioares problem or secure scalar product.

Our protocols assume the semi-honest model of computation. This model is one of the widely accepted models. We acknowledge that this model is not suitable for all situations. We also acknowledge that some of our protocols produce information leaks, but describe these risks explicitly and we describe why we believe that these leaks are innocuous.

Our protocols are implementable. In fact most of them have been already implemented. They are also practical. Moreover, we will show by experimentation with such implementations that our protocols are far more efficient than existing ones. Thus, we argue we have achieved a useful balance between privacy and practicality.

1.6 Bibliography at a Glance

Several papers have been published in the past titled privacy-preserving data mining [6, 45, 77, 96]. They are also papers that shifted the research in this area [4, 13, 36, 37, 32]. Two of data-mining core tasks are association rule mining and classification. Thus, it would be natural to expect several papers in the area of privacy-preserving association rule mining [47, 92, 66] and several as well dealing with classification [38, 67, 39].

Privacy-preserving techniques that use data perturbation methods [6, 35, 47] are also the focus of active research. There are classic references in the secure multi-party computational area which provide the origin of this field [54, 102, 55]. This remains one of the hot topics in the privacy-preserving data mining [36, 13, 37, 39, 93, 89]. Other interesting on-line references and papers are also available [14, 21, 50, 20, 23, 27, 28, 29, 30, 31, 41, 75, 81, 1, 91].

Chapter 2

Secure Multiparty Computations

In this chapter we present new SMC protocols as well as review some existing ones. General SMC protocols are the basis of privacy preservation for distributed parties. These basic protocols enable us to construct more complex protocols, to carry out more complex tasks than just computing, for instance, the scalar product of two vectors. First we introduce privacy-preserving models that exist in the literature, then we move on describing several SMC protocols.

2.1 Theory and Origin

We study collaboration between several parties that wish to compute a function of their collective databases. In fact, they are to conduct data mining tasks on the joint data set that is the union of all individual data sets. Each wants the others to find as little as possible of their own private data.

To focus the discussion on privacy-preserving collaboration and for simplicity, we will regularly use two parties Alice and Bob, in some cases three (Alice, Bob and Charles). However, if the solutions for the case of more than two or three parties differ we will make note of it.

Every record in the database is an attribute-value vector. Data partitioning is called vertically partitioned data between two parties, Alice and Bob, if one part of every vector is owned by Alice and the other part by Bob (see Fig. 2.1). In the case of more than two parties, then every party will own some part (a number of attributes) from the attribute-value vector.

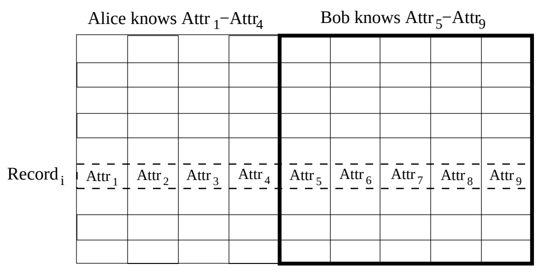


Figure 2.1: Vertically partitioned data.

There is also horizontally partitioned data, where (see Fig. 2.2) some of the records are owned by Alice and the others by Bob. Obviously, for more than two parties, every party will own some part (a number of records) from the database.

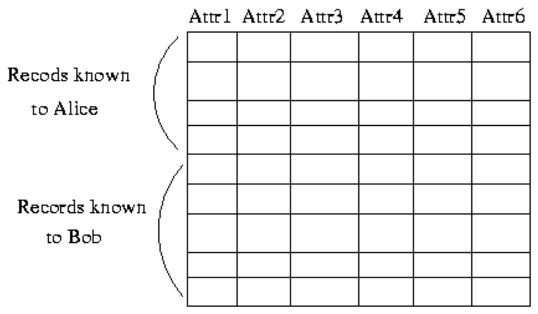


Figure 2.2: Horizontally partitioned data.

A direct and naive use of data mining algorithms on the union of the data requires one party to receive data (every record) from all other parties, or all parties to send their data to a trusted central place. The recipient of the data would conduct the computation in the resulting union. In settings where each party must keep their data private, this is unacceptable. Note that, for horizontally partitioned data, the more parties are involved, the more records are involved and the larger is the global database.

Note that, for vertically partitioned data, the more parties are involved, the more attributes are involved and the higher the dimensions of the attribute-vectors. For simplicity, we can identify each domain with one party (so the dimension m of the records is also used as the number or parties). Typically there would be strictly fewer parties than dimensions (as in Fig. 2.1 where two parties have data for 9-dimensional records). However, we consider Alice as 4 virtual parties (one for each of the columns) and Bob as 5 virtual parties each controlling one of Bob's columns. This simplifies the notation in the algorithms, and communication between two virtual parties of the same party just does not need to occur.

When the data is horizontally partitioned, for simplicity, we may assume each party owns one record only, so the number P of parties is also the number n of records. Typically there would be more records than parties (as in Fig. 2.2 where two parties have data for 9 records). However, we consider Alice as 4 virtual parties (one for each of the records) and Bob as 5 virtual parties each controlling one of Bob's records. Here again, this simplifies the notation in some of the algorithms and communication between two virtual parties just does not need to occur.

As we mentioned in Page 12 , our approach is based on the theory developed under the name of Secure Multiparty Computation. Recall that in such setting, Alice holds one input vector $\vec{x}$ and Bob holds an input vector $\vec{y}$. They both want to compute a function $f(\vec{x}, \vec{y})$ without each learning anything about the other's input expect what can be inferred from $f(\vec{x}, \vec{y})$. Yao's Millionaires Problem [102] provides the origin for SMC. In the Millionaires, Alice holds a number a while Bob holds b. They want to identify who holds the larger value (they compute if $a > b$) without either learning anything else about the other's value. The function $f(x, y)$ is the predicate $f(x, y) = x > y$. There are recent solutions for this problem. One uses a semi-trusted third party [22], whereas the others does not [63, 88]. However both later solutions [63, 88] use oblivious transfer.

Secure multi-party computation under the semi-honest model [54] has regularly been used for privacy-preserving data mining [36, 39, 93]. Here we work under the semi-honest as well, which means all parties will follow the protocol since all are interested on the results. However, all parties can use all the information collected during the protocol to attempt to discover the private data or some private values from another party.

2.2 The Semi-Honest Model

Let us emphasise the model of computation for our protocols. The semi-honest party is the one who follows the protocol correctly, but at the same time keeps information received during communication and final output, to attempt later to disclose private information from other parties. A semi-honest party is sometimes called an *honest but curious one* [64].

The SMC literature provides various formal definitions for the semi-honest model. The most common one is the following [54]:

Definition 1 *(privacy w.r.t. semi-honest behaviour): Let*

$$f : \{0,1\}^* \times \{0,1\}^* \longrightarrow \{0,1\}^* \times \{0,1\}^*$$

be a functionality, and $f_1(x,y)$ *(respectively,* $f_2(x,y)$*) denote the first (resp., second) element of* $f(x,y)$*). Let* Π *be two-party protocol for computing* f*. The view of the first (resp., second) party during an execution of protocol* Π *on* (x,y)*, denoted* $view_1^{\Pi}(x,y)$ *(resp.,* $view_2^{\Pi}(x,y)$*), is* $(x, r_1, m_1, \cdots, m_t)$ *(resp.,* $(y, r_2, m_1, \cdots, m_t)$*), where* r_1 *represents the outcome of the first (resp.,* r_2 *the second) party's internal coin tosses, and* m_i *represents the i-th message it has received. The output of the first (resp., second) party after an execution of* Π *on* (x,y) *is denoted* $output_1^{\Pi}(x,y)$ *(resp.,* $output_2^{\Pi}(x,y)$*), and is implicit in the party's view of the execution, and* $output^{\Pi}(x,y) = (output_1^{\Pi}(x,y), output_2^{\Pi}(x,y))$

- *(general case) We say that* Π *privately computes* f *if there exists a probabilistic polynomial-time algorithm, denoted* S_1 *and* S_2*, such that*

$$\begin{aligned} \{S_1(x, f_1(x,y), f(x,y)\}_{x,y\in\{0,1\}^*} &\equiv^C \{view_1^{\Pi}(x,y), output^{\Pi}(x,y)\}_{x,y\in\{0,1\}^*} \\ \{S_2(y, f_2(x,y), f(x,y)\}_{x,y\in\{0,1\}^*} &\equiv^C \{view_2^{\Pi}(x,y), output^{\Pi}(x,y)\}_{x,y\in\{0,1\}^*} \end{aligned}$$

where $\equiv^C$ *denotes computational indistinguishability by (non-uniform) families of polynomial-size circuits. Here* $view_1^{\Pi}(x,y)$*,* $view_1^{\Pi}(x,y)$*,* $output_1^{\Pi}(x,y)$ *and* $output_2^{\Pi}(x,y)$ *are related random variables, defined as a function of the* *same random execution. In particular, output*$_i^\Pi(x, y)$ *is fully determined by view*$_i^\Pi(x, y)$*.*

This definition basically says, that a computation is secure if the view of each party during the execution of the protocol can be simulated from the input and the output of that party. Thus, for a security proof, it is enough to show the existence of a simulator for each party that satisfies the above equations, and no other party notices that its partner has been replaced by the simulator. Note however, that whatever information is derived or inferred from final result cannot obviously be kept secret.

For instance, Alice, Bob and Charles may be each holding private numbers and want to calculate the average of their numbers. Say, by some SMC protocols they discovered that the average is m. Assume Charles holds c that is greater than m. This immediately discloses that at least one of the other parties holds a number less than the value m. This information was inferred from the final result and not during the execution of the protocol, thus the protocol still can be considered secure under the semi-honest model.

Theorem 1 *(Composition theorem for the semi-honest model): [54]*

Suppose that g is privately reducible to f and that there exists a protocol for privately computing f. Then there exists a protocol for privately computing g.

2.3 Output in Private Shares

Assume we are to compute some $f(a, b)$, where a belongs to Alice and b belongs to Bob. The function $f(a, b)$, for instance, could be the predicate $f(a, b) = a > b$. Again Alice and Bob do not want to disclose their private data to each other. It is possible also that their final goal is not to compute $f(a, b)$, but some $g(f(a, b))$, namely the value $f(a, b)$ is an intermediate result that would be used to compute $g(f(a, b))$. In this kind of situation it is not desirable to provide intermedate results to each other, because it could leak some information that if combined with the overall output g could reveal more than the output of g alone may have revealed. The solution is to distribute all intermediate results in private shares, then with specific techniques use these shares to obtain an overall result. Hence, for basic SMC protocols it is always desirable to have a version with private shares as well, rather than having only a version without private shares.

Consider one small example. Assume Alice has a number a, Bob has a number b and Charles has a number c, they want to know who holds the maximum value. One solution could be that, for instance, Alice securely compares her value with Bob, if she holds the greater one, she goes on to compare her value with Charles. Let's say, Charles holds the maximum value, so he is the winner. However, Charles would easily discover that Alice's value was greater than Bob's, because she was the one who compared her value with him. Thus, the execution of this protocol not only reveals who holds the maximum value but also reveals the sorted order to each party.

Now let us see how a method with private shares would work. Alice has a number a, Bob has a number b and Charles has a number c, they what to know who holds the maximum value. Alice securely compares her value with Bob, but the output is distributed into shares, namely s_a^b goes to Alice and s_b^a goes to Bob, where

$$s_a^b + s_b^a = \begin{cases} 1 & \text{if } a > b, \\ 0 & \text{if } a \leq b. \end{cases}$$

Because they both do not know who is the winner, they should compare their values with Charles as well. Thus, they will all have the following shares:

$$\text{Alice obtains} \begin{pmatrix} s_a^b \\ s_a^c \end{pmatrix} \qquad \text{Bob obtains} \begin{pmatrix} s_b^a \\ s_b^c \end{pmatrix} \quad \text{Charles obtains} \begin{pmatrix} s_c^a \\ s_c^b \end{pmatrix}.$$

Now, they apply the secure ADD VECTORS PROTOCOL (see Section 2.5) to the following vectors:

$$\text{Alice uses} \begin{pmatrix} s_a^b + s_a^c \\ s_a^b \\ s_a^c \end{pmatrix}, \text{ Bob uses} \begin{pmatrix} s_b^a \\ s_b^a + s_b^c \\ s_b^c \end{pmatrix} \text{and Charles uses} \begin{pmatrix} s_c^a \\ s_c^b \\ s_c^a + s_c^b \end{pmatrix}.$$

Alice obtains the sum in permuted order. If there were no equal numbers between a, b and c then the sum vector would be something like this (assuming Charles holds the maximum)

$$\text{sum} = \begin{pmatrix} 0 \\ 2 \\ 1 \end{pmatrix}.$$

Because of the permutation, Alice does not know who is the winner, she knows only that the permuted ID of the winner is 2, so she sends this ID=2 to Bob, because Bob was the one who permuted the vector. Bob finds the real ID, which is 3 and sends this to all.

Obviously in this case none of the parties discovers the ordering. Thus, this solution is more preferable for preserving privacy purposes.

2.4 Other Models

Three models have been proposed for assessing the privacy of protocols in privacy-preserving data mining. The most common model is the *semi-honest* model, while less common are the *malicious model* and the *weak* model. The weak model was used for many algorithms involving matrix operations and in particular, linear regression [38, 96]. In this model, security is regarded with respect to certainty. Therefore, one party is considered not to have bridged the security as long as there are an infinite number of possibilities for the values of the other parties. This model has been criticised, because it can consider secure a protocol where one party learns some information about another party's data. For example, Alice could learn that Bob's b_1 value is in a small range. While there are an infinite number of rationals (or reals) in this interval, this could provide enough precision for it to be considered a security leak. Learning or discovering an interval is discovering a distribution of the value. If the distribution has very small variance, although a large range, the security leak could be serious.

The malicious model expects both parties to behave as destructively as possible in attempting to discover data from one another. Thus, parties may supply false data, and interrupt the protocol. This model has been relegated to extreme cases, where the parties are not interested in the final result. Parties are not collaborating, but are attempting to infiltrate and perhaps damage each other.

Thus, the semi-honest model has prevailed as the most common model. The formal definition of the semi-honest model is rather technical (see Page 18) as it is the mechanism to prove security of protocols. However, we show that demonstrating that a protocol is secure in the spirit of this protocol is not beyond a clear and transparent argument (see Section 2.3). One has to demonstrate that each party does not learn anything about another party's data except what can be learned from its own data and the result. All messages received appear as random values (as if they were generated by an oracle [54]), and thus, the party could complete the protocol correctly in polynomial time even if the messages were replaced by the random values provided by the oracle. We are to assume that all parties will behave in this way, and will follow and complete the protocol with genuine interest in the results, therefore, not supplying false data that would make the result invalid.

While some protocols for vector and matrix operations have been dismissed as only secure on the weak model and not in the semi-honest model, we believe one cannot discard the merit of these protocols, specially if they are regarded as not secure in the semi-honest model by a technicality. A case in point is the protocol for scalar product that provides the output in shares [39]. This protocol is regarded as secure in the weak model sense because it requires a commodity server. We argue here that the commodity server does not contradict the spirit of the semi-honest model. We can consider the commodity server as a third party in the protocol, with empty input and empty output. The three parties would be interested in computing $f(\vec{x}, \vec{y}, \lambda) = s_A + s_B$, where the input λ of the third party (the commodity server) is empty and will not affect the output value $s_A + s_B$ discovered by Alice and Bob with respective private shares. As long as the third party does not discover anything about Alice's input $\vec{x}$, Bob's input $\vec{y}$, Alice's share s_A and Bob's share s_B, then the protocol is secure in the spirit of the semi-honest model. In fact, many times a protocol among more than two parties requires a sub-protocol in which two parties compute a value with the assistance of a third (and the protocol remains within the semi-honest model framework).

In Section 2.8 we reproduce the scalar product protocol from Du and Zhan [39] and show that neither party (including the commodity server) learns anything beyond its own input and what can be inferred from the result. Therefore, this protocol is secure in a sense stronger than the weak model and is secure in the spirit of the semi-honest model.

2.5 Review of the Add Vectors Protocol

The technique was introduced for manipulation of vector operations as the "permutation protocol" [36] and is also known as the "permutation algorithm" [93]). This protocol can provide an output in secret shares, so will discuss this as well.

In this protocol, Alice has a vector $\vec{x}$ while Bob has vector $\vec{y}$ and a permutation π. The goal is for Alice to obtain $\pi(\vec{x}+\vec{y})$; that is Alice obtains the sum $\vec{s}$ of the vectors in some sense. The entries are randomly permuted, so Alice cannot perform $\vec{s}-\vec{x}$ to find $\vec{y}$. Also, Bob is not to learn $\vec{x}$. The solution is based on homomorphic encryption for which many implementations are possible. The protocol works as follows.

Protocol 1 *1. Alice produces a key pair for a homomorphic public key system and sends the public key to Bob. We denote by $E(\cdot)$ and $D(\cdot)$ the corresponding encryption and decryption system.*

2. *Alice encrypts $\vec{x}=(x_1,\cdots,x_n)^T$ and sends $E(\vec{x})=(E(x_1),\cdots,E(x_n))^T$ to Bob.*

3. *Using the public key from Alice, Bob computes $E(\vec{y})=(E(y_1),\cdots,E(y_n))^T$ and uses the homomorphic property to compute $E(\vec{x}+\vec{y})=E(\vec{x})\times E(\vec{y})$. Then, he permutes the entries by π and sends $\pi(E(\vec{x}+\vec{y}))$ to Alice.*

4. *Alice decrypts to obtain $D(\pi(E(\vec{x}+\vec{y})))=\pi(\vec{x}+\vec{y})$.*

This can be extended to the case of $P\geq 3$ vectors, that is $P>2$ parties are involved. In this case there is no need to permute the result, because $D(\cdot)$ is known only by Alice, and Alice will get the value $E(\vec{v}_2+\cdots+\vec{v}_P)$, where $\vec{v}_i$ is the vector owned by i^{th} party. The algorithm is as follows:

Protocol 2 *1. The 1^{st} party (Alice), generates $E(\cdot)$ and $D(\cdot)$, then sends only $E(\cdot)$ to the other parties.*

2. *Then, the P^{th} party encrypts his data $E(\vec{v}_P)$ and sends it to the $(P-1)^{th}$ party.*

3. *Next, the $(P-1)^{th}$ party encrypts his data $E(\vec{v}_{P-1})$ and using the homomorphic encryption property computes*

$$E(\vec{v}_{P-1}) \times E(\vec{v}_P) = E(\vec{v}_{P-1} + \vec{v}_P)$$

   and sends this to the $(P-2)^{th}$ party.

4. *The protocol continues until Alice (the first party) will get $E(\vec{v}_2 + \cdots + \vec{v}_{P-1} + \vec{v}_P)$ and she adds her data in the same way.*

5. *As Alice owns $D(\cdot)$, she decrypts the results and sends them to all the other parties.*

Note that in Step 1, the P^{th} party does not need to permute his result because the $(P-1)^{th}$ party does not know $D(\cdot)$ to decrypt. In this case $E(\cdot)$ could be as simple as adding a random number in a sufficiently large additive field F (or X-OR with a random bit mask) and consequently $D(\cdot)$ will be subtracting the random number previously added.

One can easily notice that for $P > 2$ (when we do not use permutation) this method can be applied for adding not only vectors but matrices or just numbers as well. This, however, sometimes is called "SECURE SUM PROTOCOL" [96].

If one is intersted in the version that provides an output in secret shares, the ADD VECTORS PROTOCOL will work if we halt at Step 4 with Bob (the second party). Then the output of the protocol will be distributed in secret shares between Bob and Alice and encryption is addition of a vector R of random entries. This means Alice holds $a = v_1 - R$ and Bob holds $b = (v_2 + \cdots + v_P + R)$.

2.6 Review of the Commodity Server

For performance reasons, for instance, less communication cost, we can use the help from an extra server, the commodity server. The commodity server can also be used for preventing some information leakages that otherwise will occur. Alice and Bob can send request to the commodity server and receive data (called commodities) from the server. The commodities should not be dependent from Alice's or Bob's private data. The purpose of the commodities is to help Alice and Bob perform the desired computation. The commodity server model has been used previously in the literature [15, 39] for solving private information retrieval problems.

2.7 Review of the Scalar Product Protocol with Private Shares

There are many scalar product protocols proposed in the literature [39, 92, 64, 36]. We do not review all of them, but the one we regularly use here is the following protocol [39]. In this protocol, Alice has a vector $\vec{x}$ and Bob has another vector $\vec{y}$ (both with n elements). Alice and Bob use the protocol to compute the scalar product $\vec{x}^T \cdot \vec{y}$ between $\vec{x}$ and $\vec{y}$, such that Alice gets V_1 and Bob gets V_2, where $V_1 + V_2 = \vec{x}^T \cdot \vec{y}$ and V_2 is randomly generated by Bob. Namely, the scalar product of $\vec{x}$ and $\vec{y}$ is divided into two secret shares. It is a protocol that produced private shares. The computation is performed in the domain of the reals $\Re$ and by *Theorem 4.1* in [39] neither Alice nor Bob can learn each other's private data.

Protocol 3 *(Scalar Product Protocol with Shares)*

1. *The commodity server generates two random vectors $\vec{\Psi}$ and $\vec{\Pi}$ of size n, and lets $r_a + r_b = \vec{\Psi}^T \cdot \vec{\Pi}$, where r_a (or r_b) is a randomly generated number. Then the server sends $(\vec{\Psi}, r_a)$ to the first party (Alice). It send $(\vec{\Pi}, r_b)$ to the second party (Bob).*

2. *Alice computes a perturbed version $\hat{\vec{x}} = \vec{x} + \vec{\Psi}$ of its vector and sends $\hat{\vec{x}}$ to Bob.*

3. *Bob also perturbs its vector with the random vector provided by the commodity server $\hat{\vec{y}} = \vec{y} + \vec{\Pi}$ and sends it to Alice. Thus, Alice obtains the*

vector $\vec{y} + \vec{\Pi}$.

4. *Bob generates a random number V_2, and computes $\vec{y}^T \cdot \hat{\vec{x}} + (r_b - V_2)$. He sends this result to Alice.*

5. *Alice adds $r_a - \vec{\Psi}^T \cdot (\vec{y} + \vec{\Pi})$ to the value received from Bob and calls it V_1. This is $V_1 = r_a - \vec{\Psi}^T \cdot (\vec{y} + \vec{\Pi}) + \vec{y}^T \cdot \hat{\vec{x}} + (r_b - V_2) = r_a + r_b - \vec{\Psi}^T \cdot \vec{\Pi} - \vec{\Psi}^T \cdot \vec{y} + \vec{y}^T \cdot (\vec{x} + \vec{\Psi}) - V_2 = \vec{y}^T \cdot \vec{x} - V_2$.*

Let us remark first that the above description enables us to show that knowledge of $n-1$ entries on Bob's vector $\vec{y}$ by Alice is insufficient for Alice to learn the remaining entry. There will still be two unknowns (V_2 being one) in the equation $V_1 + V_2 = \vec{x}^T \cdot \vec{y}$ for Alice[1].

The second point corresponds to the fact that Bob, Alice and the commodity server do not learn anything about each other's data. Clearly the commodity server does not learn anything since it never receives any messages. Bob receives a perturbed vector from Alice, with all entries appearing random. Symmetrically, Alice receives Bob's vector completely masked by random values and thus these also appear as totally random values [2]. Bob generates his own random V_2 and will not receive anything else from Alice; therefore he cannot learn anything about Alice's data. Although Alice receives the additional value $\vec{y}^T \cdot \hat{\vec{x}} + (r_b - V_2)$ from Bob, $\vec{y}^T \cdot \hat{\vec{x}} + r_b$ is totally masked by the random value V_2. Alice cannot learn anything either.

Note also that if one want to compute the scalar product without private shares Bob need only to set $V_2 = 0$. As authors of this protocol have remarked in the original paper [39] (page 6), this does not allow Alice to learn any of Bob's private data. Furthermore, under the semi-honest model, where Bob is not a malicious party, Bob does not learn anything either. Thus, this protocol can be used for computing the secure scalar product where the answer goes only to one of the parties.

The communication cost of this protocol is $4n$, which is 4 times more expensive than the *DNSP* cost of a two-party scalar product (the *DNSP* cost of a scalar

[1]This ensures that the division protocol in Sec. 5.1.1 is secure

[2]In practice, these values could range over a very large field F, and display a uniform distribution on such a field; making impossible for either party even to learn their magnitude.

product is defined as the cost of computing the product of $\vec{x}$ and $\vec{y}$ without the privacy constraints, namely one party just sends its data to the other party). The cost can be reduced to $2n$ because the vectors $\vec{\Psi}$ and $\vec{\Pi}$ are randomly generated by the commodity server, namely the commodity server can send just the seeds (numbers of constant size) to Alice and Bob, and the seeds can be used to compute the random vector. Other solutions to the scalar product protocol have been proposed before [13, 92] and both of these solutions provide privacy preservation without using a third party. In practice the communication and computation costs is more expensive, if one uses a solutions without commodity server. They have larger constants under the $O(n)$ complexity, where n is the size of the vectors.

2.8 New Scalar Product Protocol

In this section we propose a new simple scalar vector protocol which is based on the Add Vectors Protocol (see Section 2.5). This protocol is very simple and it is easy to implement. Depending on the domain and encryption it is also very efficient.
In this protocol again, Alice has a vector $\vec{x}$ and Bob has another vector $\vec{y}$ (both with n elements). Alice and Bob use the protocol to compute the scalar product $\vec{x}^T \cdot \vec{y}$ between $\vec{x}$ and $\vec{y}$, such that Alice gets $\vec{x}^T\vec{y}$.
Note that

$$2x_iy_i = (x_i - y_i)^2 + x_i^2 + y_i^2,$$

thus the protocol works as follows:

Protocol 4 *(New Scalar Product Protocol)*

1. *Alice and Bob apply the* Add vectors protocol *for Alice to obtain $\pi_0(\vec{x} - \vec{y})$, were π_0 is a permutation generated by Bob.*

2. *Alice and Bob apply the* Add vectors protocol *again for Alice to obtain $\pi_0(\vec{x}^2 + \vec{y}^2)$, where π_0 is the same permutation generated by Bob in step 1.*

3. *Now Alice is able to compute the following vector*

$$\pi_0 \begin{pmatrix} 2x_1y_1 \\ \vdots \\ 2x_ny_n \end{pmatrix} = \pi_0 \begin{pmatrix} -(x_1-y_1)^2 \\ \vdots \\ -(x_n-y_n)^2 \end{pmatrix} + \pi_0 \begin{pmatrix} x_1^2+y_1^2 \\ \vdots \\ x_n^2-y_n^2 \end{pmatrix}.$$

4. *The only thing left for Alice is to sum up the element of the obtained vector and divide by* 2 *to obtain the scalar product*

$$\vec{x}^T \cdot \vec{y} = \sum_{i=1}^{n}(x_iy_i) = \frac{1}{2}\sum_{i=1}^{n}\left(-(x_i-y_i)^2 + x_i^2 + y_i^2)\right).$$

Alice learns all the values $2(x_iy_i)$ from the information collected during the protocol's execution. However, this is not enough for Alice to discover any of Bob's private data, because of the permutation applied by Bob. Essentially, this protocol's security is the same as the ADD VECTORS PROTOCOL.

Note that the encryption used in the ADD VECTORS PROTOCOL can be as simple as adding random vector consisting of the same random number. Alice then adds this random vector to her vector and sends the results to Bob, whereas Bob only adds his vector to the sum vector received and performs a permutation before he sends it back to Alice. However, this solution should be handled carefully, because if Bob knows any of the coordinates in Alice's vector, then it immediately discloses the random number and consequently all of Alice's vector.

2.9 New Protocol for the MAXIMUM VALUE IN THE SUM OF VECTORS.

We will look at the two party case separately from those of three or more parties since these require slightly different solutions.

Two Party Case: Consider two attribute vectors $\vec{x}$ and $\vec{y}$, where only Alice knows $\vec{x}$ and only Bob knows $\vec{y}$. (where $\vec{x} = (x_1, \cdots, x_n)^T$ and $\vec{y} = (y_1, \cdots, y_n)^T$). The goal is to obtain $\max_i(x_i + y_i)$. There is a variant that obtains the index i_0 for which $x_i + y_i$ is maximum. However, the protocol should never reveal both, because then both parties find a value for the other. For example, Alice would find $\max_i(x_i + y_i) - x_{i_0} = y_{i_0}$. Even if the maximum value is the only outcome revealed, and the protocol is ideal, each party learns the value $x_{i_0} + y_{i_0}$ of the maximum, and thus, each party can subtract its data vector to find n values among which one value is from the other party. For example, Alice can compute $x_{i_0} + y_{i_0} - x_i$ (for $i = 1, \ldots, n$) and she finds n values one of which must be y_{i_0} owned by Bob.

Bob starts by generating a random value $r > 0$ and using $\vec{y}+r = (y_1+r, \ldots, y_n+r)^T$ in the ADD VECTOR PROTOCOL from Section 2.5. This provides Alice with $\pi(\vec{x} + \vec{y} + r)$; that is, Alice obtains the sum vector in permuted order, which will suffice for Alice to find the maximum and inform Bob. If the maximum value is the outcome sought, Alice provides only the value $x_{i_0} + y_{i_0} + r$. Bob will subtract r and pass $x_{i_0} + y_{i_0}$ to Alice. If the index where the maximum lies is sought, then Alice provides only the index where she found the maximum and Bob applies π^{-1} to broadcast the index. Note that Alice cannot learn which of the coordinates provided the maximum value, until Bob broadcast it. Note however, only when the maximum is sought, Alice learns the random number r and all the values in the entries of $\pi(\vec{x} + \vec{y})$, but this is not enough to learn any of Bob's private data.

The Multiparty Case ($P > 2$): Here, direct use of our extension to the ADD VECTORS PROTOCOL will reveal slightly more information than we would like to; namely, Alice obtains the sum of the vectors without the effect of the permutation (see Subsection 2.5). She is able not only to find the maximum, but the index i_0 which holds the maximum of $x_i + y_i$ as well. So, Alice will know a bit more than the others. Therefore, we will improve this protocol. The proposed protocol works as follows. Assume there are $P > 2$ parties and every party has a vector $\vec{v}_i$.

Protocol 5 *1. The 1st party (Alice), generates a random vector $\vec{R} = (r_1, \ldots, r_n)$ and sends it to the P^{th} party.*

2. Then, the P^{th} party adds his vector $\vec{v}_P$ to the random vector received from Alice and sends the sum to the $(P-1)^{th}$ party.

3. The protocol continues until Bob (the second party) receives the sum $\vec{S}_b =$

$\vec{v}_3 + \cdots + \vec{v}_P + \vec{R}$

4. *Alice and Bob use our finding the* MAXIMUM VALUE IN SUM OF VECTORS*(or the index) for* $P = 2$ *with the vectors* $\vec{S}_a = \vec{v}_1 - \vec{R}$ *owned by Alice and* $\vec{S}_b = \vec{v}_3 + \cdots + \vec{v}_P + \vec{R} + r$ *owned by Bob (Bob randomly generates* r*), which allows Alice to obtain* $\pi(S_a + S_b) + r$*; that is the sum (translated by* r*) of the vectors in permuted order.*

5. *Alice finds and passes the maximum value to Bob who subtracts* r *and broadcasts the result to the other parties (or alternatively, if the index is sought, Alice will pass the index to Bob who will apply* π^{-1} *and broadcasts the index to all parties).*

Lemma 1 *Under the semi-honest model, for* $P > 2$*, the version finding* MAXIMUM VALUE IN THE SUM OF VECTORS *protocol does not allow any party to learn any party's value nor the index for where such maximum lies.*

Proof: The lemma holds in the multiparty ($P > 2$) case because the random vector generated by Alice masks the data from the 3^{rd} party to the P^{th}. For Alice and Bob (the first and second parties), the protocol is secure as per the two party case, except that this time, even if Alice subtracts all the values in its vector from the maximum value, she only obtains partial sums and not a value from the other parties. □

A similar result applies to the version that broadcasts the index but not the maximum value. In the literature, there are several algorithms to find privately not the maximum value but the index of the maximum value in a sum of vectors [43, 93]. Such algorithms are very costly; some require a theoretical construction of a combinatorial circuit, that is hard to implement [93] while others [43] required P iterations of the ADD VECTORS PROTOCOL and a matrix of random values, plus m iterations of a division protocol. Those algorithms become slightly more efficient if they allow one party to find all the sums. Note that our index version of the algorithm does not allow this to happen (only a translation of the sums is found by Alice).

2.10 Yao's Two Millionaires Problem - New Solution with Private Shares

One advantage of the "shares" theoretical result is that one can easily decompose the result $f(\vec{x}, \vec{y})$ into a share s_A for Alice and a share s_B for Bob, so that $s_A + s_B = f(\vec{x}, \vec{y})$, but neither party can find $f(\vec{x}, \vec{y})$ from their share. This allows us to use a protocol for one task (like Yao's comparison of two values) in a larger protocol (e.g. sorting). This usage of a protocol as a subroutine in another enables construction of more complex and secure protocols, but transmits the impracticality of the generic "shares" result further.

We present here a practical solution to Yao's millionaires problem, but provide the output in shares. Recall that here Alice holds a and Bob holds b, but then, after the protocol, they do not share knowledge of the output ($a > b$?), but the output for Alice is r_a and for Bob r_b, where

$$r_a + r_b = \begin{cases} 1 & \text{if } a > b, \\ 0 & \text{if } a \leq b. \end{cases}$$

There are several algorithms [67, 87, 94] that invoke a subroutine for Yao's comparison with shares, and all of them rely on the circuit evaluation generic "shares" theoretical solution [54]. Hence, they seem far for implementation. We present here a practical and inexpensive solution that we apply in our algorithms. This solution can also alleviate the implementation issues for the above mentioned protocols. It uses a third semi-trusted party (also commonly used in the privacy-preserving literature [39] [3]). It is also common that when there are more than 2 parties, they take turns performing the role of the third party for two others [93]).

Checking whether $a > b$ is the same as checking whether $a + (-b) > 0$. In our protocol, we will use a semi-trusted third party.

Protocol 6 *Solution for Yao's millionaires problem with private shares.*

1. *The third party generates a random number R_a and sends R_a to Alice.*

[3]This may seem a strong assumption, but is not rare in reality. An on-line auction is an example of such a party, because buyers and sellers assume that the auctioneer is non-colluding.

2. *Alice generates a random number R, where $R \in \Re \backslash \{0\}$ and sends $(R, a \cdot R + R_a)$ to Bob.*

3. *Bob adds $-b \cdot R$, and sends $(a-b) \cdot R + R_a$ to the third party.*

4. *The third party subtracts R_a and checks whether $(a-b) \cdot R > 0$.*

5. *The third party generates two pairs of values (r_a^0, r_b^0) and (r_a^1, r_b^1), where $r_a^0 + r_b^0 = 0$ and $r_a^1 + r_b^1 = 1$. If $(a-b) \cdot R < 0$, then the third party sends (r_a^0, r_a^1) to Alice and (r_b^0, r_b^1) to Bob. If $(a-b) \cdot R \geq 0$, then the third party sends (r_a^1, r_a^0) to Alice and (r_b^1, r_b^0) to Bob.*

6. *If $sign(R) = 1$ (i.e. $R > 0$) Alice and Bob use as their shares the first value in the pair received from the third party. While if $sign(R) = -1$, each picks as their share the second number in the pair received from the third party.*

Let us see what every party obtains from this protocol.

1. Alice obtains R, R_a and the set $\{r_a^1, r_a^0\}$. However, she has no way of telling which one is which in the set $\{r_a^1, r_a^0\}$.

2. Bob obtains R, the set $\{r_b^1, r_b^0\}$, and $a \cdot R + R_a$. Again, Bob cannot tell which one is which on the set $\{r_b^1, r_b^0\}$.

3. The third party gets R_a, r_a^1, r_a^0, r_b^1, r_b^0, and $(a-b) \cdot R$.

Note that, the third party does not know the value $a-b$, since $sign(R) \in \{-1, 1\}$.

A small concern is that in the case $a = b$, the third party learns that $a = b$, although it does not learn anything else (the values a and b remain inaccessible to the third party). Note that our Yao's comparison makes the party that supplies the second argument in the Yao comparison a winner when values are equal. This implements implicitly and effectively a comparison where a later indexed party is a winner among parties with equal values. It is possible to increase the uncertainty in the third party by Alice and Bob using the third party with additional dummy Yao's tests.

Neither Alice nor Bob learns anything. Alice receives no messages or values from Bob and she receives values equivalent to random numbers from the third party. Similarly, Bob receives $(R, a \cdot R + R_a)$, this is the only message from Alice and since R_a is a random number, he cannot infer a. Neither can he infer the choice by the third party.

A security proof by simulation [54] is obvious. Bob just needs to get a random value for $a \cdot R + R_a$ and in polynomial time adds $-b \cdot R$ with output given by the third party. Alice's simulation is even more trivial since she receives nothing from Bob. The output is also from the third party.

We have implemented the sign-based solution presented above. Implementation requires some adaptation, since the value $R \in \Re \setminus \{0\}$ generated by Alice cannot be any non-zero real. However, all implementations of a solution to Yao's Millionaires problem assume that the values of a for Alice and b for Bob are in a large but bounded interval; that is $a, b \in [0, M]$ where M is very large (and known to both Alice and Bob). Even those solutions that do not provide the answer with shares require this and typically assume further that a and b are integers in a very large field. Therefore, our implementation of the sign-based solution also assumes that a and b are integers with $a, b \in [0, M]$, and M is known to the implementer. Since Bob will learn R and $aR + R_a$, the value R_a produced by the commodity server must mask aR (otherwise Bob may learn some bits about a). Since aR can have as many bits as $\log_2 M + \log_2 R$, we typically let Alice choose R so that $|R| > M$ and the trusted party chooses $R_a > 2M$. For example, Alice can choose R uniformly in $[-2M, M) \cup (M, 2M]$.

After this adjustment, the implementation does not reveal information to Alice or Bob about each other's value. However, the third party does learn $(a - b)R$ and because $|R|$ is bounded (in fact, the distribution of $|R|$ will be known to the third party), this party learns approximations to $|a - b|$. That is, the third party will not learn who between Alice and Bob holds the larger value, but will gain an idea on the gap that exists between the two. We regard this leak of information in the sign-based protocol as innocuous, given the estimates of $|a - b|$ have relative error, and the other advantages it provides.

As we mentioned earlier (see Section 1.5) the literature has a general solution for all polynomially bound computations [55]. These general solutions require f to be explicitly represented as a Boolean circuit of polynomial size. It has been shown [78] that a solution for the Yao's millionaires problem with circuit evaluation techniques would require a total 254 circuit gates, out of which 64 are inputs and 2 are outputs.

First, our solution is the first implementable protocol for Yao's millionaires problem with shares (we have a C++ implementation over sockets). Second, it is far more efficient that other solutions to Yao's millionaires problem, even without shares. Cachin's solution [22] [4] is linear on $\log_2(a+b)$; but, it also requires a semi-trusted party and very heavy cryptographic machinery. A solution that has been demonstrated to be efficient enough for ETHERNET networks [63] requires quadratic time and quadratic number of messages on $\log_2(a+b)$ and also as many oblivious transfers as $\log_2(a+b)$. Other practical protocols [14] also require $O(log_2(a+b))$ rounds of oblivious transfer. The solution [88] that has communication complexity of 3, still uses oblivious transfer. Moreover the computational complexity of this solution [88] is quite expensive (5 modular exponentions and $O(2^n)$ comparisons and XOR operations) compared to our solution. Oblivious transfer implementation usually requires at least two messages with a key each. The sign-based protocol requires three messages with a $\log_2(a+b)$ (one from the semi-trusted party to Alice, one from Alice to Bob and one from Bob to the untrusted party). The last round of messages have constant size 2 bits. So we have linear complexity on the size of the message (with a constant value 2) and constant number of messages. This analysis is so overwhelmingly clear in favour of the sign-based protocol that we feel direct comparison to any other implementation of a solution to Yao's millionaires to be unnecessary. The sign-based protocol requires so little machinery that it is also much easier to implement.

[4] We let $\log_2(a+b)$ denote the number of bits of $a+b$.

2.11 New Division Protocol with Private Shares

The SMC division protocol [36], does not provide an answer with shares, it gives an answer to one party only. Recently a number of protocols that can carry out secure division have been developed [12]. We are particularly interested in 2-party division protocol that can provide output with secret shares. Thus, the protocol DIV3 [12] can be used for this task. The protocol DIV3 is based on homomorphic encryption. However we provide an alternative protocol without any cryptographic techniques.

Protocol 7 *In the division protocol, Alice holds (a_1, a_2) and Bob holds (b_1, b_2). The goal is for Alice to obtain A and Bob B, where*

$$A + B = \frac{a_1 + b_1}{a_2 + b_2}.$$

1. *Alice produces a random number r_1 and Bob produces a random number r_2.*

2. *Using the* SCALAR PRODUCT PROTOCOL WITH PRIVATE SHARES [5]*, but providing an answer to one party only, Alice can obtain*

$$r_2(a_2 + b_2) = (a_2, 1)^T \cdot \begin{pmatrix} r_2 \\ r_2 b_2 \end{pmatrix}$$

and by using the same protocol the other way around [6] Bob can obtain

$$r_1(a_2 + b_2) = (b_2, 1)^T \cdot \begin{pmatrix} r_1 \\ r_1 a_2 \end{pmatrix}.$$

3. *Alice and Bob again perform the scalar product protocol with private shares (see Section 2.7), which provides an answer in private shares A and B, with*

[5]Here Bob sets his private share V_2 equal to 0 (see Section 2.7)

[6]Here Alice sets her private share V_2 equal to 0 (see Section 2.7)

the following vectors

$$\begin{aligned} A + B &= \left(r_1 a_1, \frac{1}{r_2(a_2+b_2)}\right)^T \cdot \left(\begin{array}{c} \frac{1}{r_1(a_2+b_2)} \\ r_2 b_2 \end{array} \right) \\ &= \frac{r_1 a_1}{r_1(a_2+b_2)} + \frac{r_2 b_2}{r_2(a_2+b_2)} = \frac{a_1+b_1}{a_2+b_2}. \end{aligned}$$

This terminates the presentation of new practical protocols that would be used as building blocks for more sophisticated protocols. The ultimate goals are protocols for clustering and regression in the privacy-preserving context, but first these SMC protocols will be used for calculation of various metrics and k-Near Neighbours queries.

Chapter 3

Metrics and k-Near Neighbours Queries

As we mentioned earlier (see Section 1.3) associative queries are very important for information retrieval and are used for carrying out various tasks in data mining. This wide variety of data mining tasks, for which k-NN or range queries is a fundamental operation, has recently prompted approaches to privacy preserving k-NN [8, 67, 87, 94, 95]. However, these approaches do have some serious shortcomings. For example, the suite of privacy preserving algorithms [95] to create a privacy preserving version of Fagin's *A0* algorithm [48] proved costly (see Section 4.3.4) although the authors argued that disclosure of some additional information (the union of all items in a set required to get k intersecting items) was necessary for reasonable efficiency. Other limitations have commonly been the need to compute all pairs of distances [94], to have the query-point public [67], or to deal with horizontally partitioned data [8, 87].

Associative queries are based on different metrics and disscussion about k-NN or range queries cannot be separated from proper disscussion of the metrics. Hence in order to present the construction of associative queries in the privacy context, we will focus on privacy-preserving metrics first.

3.1 Metrics for Horizontally Partitioned Data

3.1.1 Euclidian Metric

Here Alice has again a vector $\vec{x}$ while Bob has vector $\vec{y}$ and the goal is computation of the Euclidean distance between these vectors [94] while preserving privacy. Alice replaces each component x_i with three components $x_i^2, -2x_i, 1$ while Bob replaces each y_i component with $1, y_i, y_i^2$. The scalar product for these three components will then be $x_i^2 - 2x_iy_i + y_i^2 = (x_i - y_i)^2$. In general

$$\begin{aligned}\sum_i (x_i - y_i)^2 &= (\sum_i x_i^2, -2x_1, ..., -2x_n, 1)^T \cdot \\ &\quad (1, y_1, ..., y_n, -\sum_i y_i^2),\end{aligned}$$

and thus the Euclidean distance between two vectors can also be expressed as a scalar product of two vectors. Hence one could use the SCALAR PRODUCT PROTOCOL WITH PRIVATE SHARES (see Section 2.7), to compute a securely Euclidean distance. The result is two pieces of information V_1 and V_2, with V_1 going to Alice, V_2 going to Bob and $\sum_i (x_i - y_i)^2 = V_1 + V_2$. Thus, this metric can be computed into private shares.

3.1.2 Chessboard or L_∞ Metric

It is known that the Euclidean distance is not robust in higher dimensions due to the many terms involved in the sum and the potential that few values are magnified by squaring them. In other words, distinguishing what is close and what is far becomes very difficult. This is called the *curse of dimensionality.* To ameliorate this unpleasant fact, other metrics may be needed. We explore here the chessboard distance which is defined as

$$L^\infty(\vec{x}, \vec{y}) = \max(|x_1 - y_1|, ..., |x_n - y_n|).$$

Note that

$$(|x_1 - y_1|, \ldots, |x_n - y_n|)^T = |\vec{x} - \vec{y}| = |\vec{x} + (-\vec{y})|.$$

In order to compute the chessboard distance securely we can use our protocol for finding the maximum value in a sum of vectors for two parties. Note that Alice learns $\pi(|x_i - y_i|)$ which is not leeking information about Bob's values.

3.1.3 Chessboard or L_∞ Metric with Shares

Although the private chessboard distance (Section 3.1.2) calculation by using the ADD VECTORS PROTOCOL (Section 2.5) works fine when the overall output sought is the chessboard distance. As we mentioned earlier (see Section 2.3) if one wants to use it as an intermediate calculation for some other protocols, like k-NN queries, it is more preferable not to disclose all the intermediate results (distances). It is better to keep them secret (e.g. with private shares) and only disclose the overall result (the k-NNs).

Thus, we need a chessboard distance with shares that we can use as an intermediate protocol. As we have mentioned already

$$L^\infty(\vec{x}, \vec{y}) = \max(|x_1 - y_1|, ..., |x_n - y_n|) = |\vec{x} + (-\vec{y})|.$$

where Alice has vector $\vec{x} = (x_1, ...x_n)$ and Bob has vector $\vec{y} = (y_1, ...y_n)$. We suggest that Bob generates a random number and adds it to all y_i. Then Alice and Bob apply the ADD VECTORS PROTOCOL with Alice holding $\vec{x}$ and Bob holding $\vec{y}+r$. The result $\pi(\vec{x}+\vec{y}+r)$ will go to Alice and Alice can calculate the maximum, say x_0+y_0+r. Now we have the chessboard distance with shares; that is, Alice holds $x_0 + y_0 + r$ and Bob holds $-r$, where $L^\infty(\vec{x}, \vec{y}) = x_0 + y_0 + r - r$.

3.1.4 Other Metrics

Other metrics that weight attributes differently can also be computed securely using a strategy similar to the one above (for example, variants like $(\vec{x}-\vec{y})^T W(\vec{x}-\vec{y}) = \sum_i w_i(x_i - y_i)^2$ where W is a diagonal matrix of weights known to all parties). It should now be clear that it is also possible to compute securely the cosine metric $\frac{\vec{x}^T \cdot \vec{y}}{(\|\vec{x}\|\|\vec{y}\|)}$ as the scalar product of $\left(\frac{\vec{x}}{\|\vec{x}\|}\right)^T \cdot \left(\frac{\vec{y}}{\|\vec{y}\|}\right)$.

3.2 Metrics for Vertically Partitioned Data

3.2.1 Minkowski Metrics

In this section we will show how to carry out the secure computations of all *Minkowski* metrics (among them, the Euclidean metric) and also of the L_∞ distance. Also, we expect that in applications of vertically partitioned data, the domains of the attributes may be very diverse, including many units, and types, some being categorical and others ordinal or numerical. It is well known that in Instance-based Learning or k-NN classification, typically the metric is a weighted convex combination of metrics for each attribute domain. The discussion of algorithms for *Minkowski* and L_∞ will enable algorithms for combinations of local metrics into a global metric. Equipped with this, we can show data structures for associative queries suitable for privacy-preserving algorithms. Note here that in order to give a more formal treatment we do not assume the existence of virtual parties[1].

Obviously, if all parties know the r-th *Minkowski* distance $M(\vec{p}, \vec{q})$ between two vectors $\vec{p}$ and $\vec{q}$ in the database, they will also know the value $[M(\vec{p}, \vec{q})]^r$ by each raising the *Minkowski* distance to the r-th power. Conversely, if the parties find $[M(\vec{p}, \vec{q})]^r$, they can take r-th roots and find the desired distance value. Since the i-th party knows a range of the attributes of the vectors $\vec{p}$ and $\vec{q}$, it is not hard to see that

$$[M(\vec{p}, \vec{q})]^r = \sum_{i=1}^{m} \sum_{\substack{\text{attr. known} \\ \text{to party } i}} \left(\begin{array}{c} j\text{-th attr. in } \vec{p} \\ \text{owned by } i \end{array} - \begin{array}{c} j\text{-th attr. in } \vec{q} \\ \text{owned by } i \end{array} \right)^r .$$

Letting v_i be the r-th *Minkowski* distance of those attributes known to the i-party, then $[M(\vec{p}, \vec{q})]^r = v_1^r + \cdots + v_m^r$, and the problem reduces to finding the value of the sum of values distributed among m parties (each contributes the knowledge of the r-th *Minkowski* distance raised to the power r in the projection that they own). This can be carried out essentially by the ADD VECTORS PROTOCOL by

[1]This assumption can still be made. It will not harm anything.

assuming the vectors hold only one coordinate [2]. However we will give a formal definition of the protocol, which will help us to focus our discussion on this particular case.

Protocol 8 *Add the m values among the $m \geq 3$ parties.*

1. *The first party (Alice) generates a random number R and passes it to the m-th party (this is like an XOR mask of random bits).*
2. *The m^{th} party adds its value v_m^r to the random number R and passes the result to the $(m-1)$ party.*
3. *For $i = m-1$ down to 2, the $(i-1)^{th}$ party adds v_i^r to the value received and passes the result to the $(i-1)$-th party.*
4. *The protocol continues until Alice (the first party) gets $(v_2^r + \cdots + v_{m-1}^r + v_m^r + R)$, then she adds her value v_1^r and subtracts the random number R. She then takes the r-th root and announces the result to all parties.*

Note that if we halt at step 3 with Bob (the second party), then we have a **protocol for the** *Minkowski* **distance with shared values** between Bob and Alice. This means Alice holds $a = v_1^r - R$ and Bob holds $b = (v_2^r + \cdots + v_m^r + R)$ where $[M(\vec{p}, \vec{q})]^r = a + b$. In some applications (in particular, k-NN queries) the calculation of the metric is an intermediate step. Although distance values may be considered innocuous information, it is more acceptable to use private shares as this adheres to the ideal principle of SMC where parties learn only what is implied by the final output. In fact, we can use encryption modulo a field F so that the shares s_a of Alice and s_b of Bob are such that $s_a + s_b = [M(\vec{p}, \vec{q})]^r$ mod F. In a k-NN query, only the ids of the k records is the ideal answer and it is more preferable than all the parties learning some exact values of the metrics to the query vector.

The *Minkowski* distance protocol with shares is trivial in the case $m = 2$ parties, since in this case, each party uses its projected metric value as its share and they

[2]Note that number of parities must be more than two. Since we have only one coordinate, the permutation will not be used and this will immediately discloe the private value of the second party to the first party.

exchange no messages at all. In the core operations for the *SASH* (or similar data structures) rather than being interested in $dist(\vec{p}, \vec{q})$ itself, the question is whether "$dist(\vec{p}, \vec{q}) < dist(\vec{p}, \vec{r})$?" In the case of the *Minkowski* distance, this is also equivalent to asking "is $[M(\vec{p}, \vec{q})]^r - [M(\vec{p}, \vec{r})]^r < 0$?". For additional privacy, then, it is better if each party contributes the difference of its projections. That is, let v_i^r be the *Minkowski* distance (raised to the r-th degree) between $\vec{p}$ and $\vec{q}$ in the projection owned by the i-th party, and let u_i^r be the *Minkowski* distance of r-th degree between $\vec{p}$ and $\vec{r}$ in the projection owned by the i-party. Then, to answer the question we compute the sum of the m values $(v_i^r - u_i^r)$ owned distributively by the m parties and each party then can check where the sum stands relative to zero. SMC of the Euclidean distance is achieved by the above *Minkowski* algorithm in the case $r = 2$. Note also that if we considered the "shares" version of the protocol and $dist(\vec{p}, \vec{q}) = s_a + s_b$ and $dist(\vec{p}, \vec{r}) = s'_a + s'_b$ (with s'_a and s_a known by Alice and s_b, s'_b by Bob), we can still ask "$dist(\vec{p}, \vec{q}) < dist(\vec{p}, \vec{r})$?" as a Yao millionaires comparison since $s_a + s_b < s'_a + s'_b$? is also $(s_a - s'_a) + (s_b - s'_b) < 0$? with $(s_a - s'_a)$ known to Alice and $(s_b - s'_b)$ known to Bob.

Theorem 2 *If three* [3] *or more parties use the Minkowski distance protocol, no party learns other parties' private data represented as an attribute in the feature vector. If the protocol is the shared distance version, even with two parties, no party learns any information.*

Proof: The protocol calculates the distance between $\vec{p} = (p_1, \cdots, p_m)$ and $\vec{q} = (q_1, \cdots, q_m)$. During the calculation each party P_l $(l = 2, \cdots, m)$ obtains

$$S_l = (p_{l+1} - q_{l+1})^r + \cdots + (p_m - q_m)^r + (p_1 - q_1)^r + R$$

where R is a random number produced by the 1st party. Thus, because R is random, for party P_l it is impossible to learn any value from $(p_{l+1} - q_{l+1})^r$ up to $(p_m - q_m)^r$ and $(p_1 - q_1)^r$.

In the case of the first party $(l = 1)$, it obtains the actual distance $[M(\vec{p}, \vec{q})]^r$ by subtracting R and taking $r - th$ root from the sum. Here, because several terms

[3]In the two party case, where each party holds only one attribute, the knowledge of the distance by some party immediately results in the disclosure of the other parties' private data. But this cannot be avoided even in the ideal case.

are involved in the sum, the first party cannot learn any attribute p_i or q_i, where $i = 2, \cdots, m$.

The case for shares follows by the discussion above. $\square$

Algorithms which do not ensure some level of privacy will not be considered when privacy is needed. However, researchers compare secure and non-secure versions to identify the overhead of privacy-preserving algorithms over a distributed non-private setting (*DNSP*). We show that for our protocols and algorithms, the cost is essentially as for a distributed non-private setting where parties would still need to incur local calculations and communication costs between them or to a central party.

For the *Minkowski* metrics, the main cost is the communication cost, which is clearly affordable. In fact any other local cost (computing local projections of the metric) would also be performed in a *DNSP*. The communication cost is also comparable to the cost in a *DNSP*. The local cost for generation and masking with a pseudo-random number is totally subsumed with the cost of the local metric. Moreover, there are usually fewer parties than dimensions; thus, passing a value among the m parties to compute a global metric is usually well within the order of cost of computing the global value in a *DNSP*.

3.2.2 Chessboard or L_∞ Metric

While *Minkowski* metrics combine the discrepancy in each attribute domain with a sum there is also the alternative of selecting the the maximum difference as the overall metric. This leads to the L_∞ metric (some researchers prefer the name "Chessboard distance"). Recall that this metric is defined as

$$dist(\vec{x}, \vec{y}) = \max(|v_1^x + (-v_1^y)|, ..., |v_m^x + (-v_m^y)|)$$

where $\vec{x} = (v_1^x, \cdots, v_m^x)^T$ and $\vec{y} = (v_1^y, \cdots, v_m^y)^T$ are two vectors).

Note that if $\vec{x}$ and $\vec{y}$ are owned by several parties on vertically partitioned data, each party can identify the largest absolute difference $v_i = |v_i^x + (-v_i^y)|$ in its projection. So the problem reduces to which of the m parties has the largest value (thus, from now, we assume party i holds $|v_i^x + (-v_i^y)|$ and the vectors have dimension m).

A first approach will use Yao's protocol (without shares) as a subroutine to deploy a finding maximum algorithm based on $m-1$ comparisons. For example, for $i = 1$ to $m-1$ compare the maximum found in the i-th first parties with the $(i+1)$ party. While this works well, the $(i+1)$-th party must interact using a protocol for Yao comparisons with the holder of the maximum among the first i parties (thus, learning who holds the maximum and has so far won the comparisons). So this approach is not secure in the ideal sense of the semi-honest SMC since additional information besides the maximum among all m entries is leaked. Again, this information leak may be considered innocuous by some, and in that case, parties may use this proposed approach. However, we now present an approach that with some additional machinery is fully secure in the semi-honest model and it is practical. The additional machinery is how to compare two numbers and distribute the output into shares (Section 2.10).

Protocol 9 *Find maximum value with shares.*
The protocol starts with each party comparing its value with every other party. Note here that, the <Alice vs Bob> comparison is not the same as the <Bob vs Alice> comparison. For instance, if Alice compares with Bob and it happens to be that Alice's value is smaller than Bob's, then they will have shares C_{AB}^A and C_{AB}^B, where $C_{AB}^A + C_{AB}^B = 0$[4], but if Bob compares with Alice the shares should add up to one. However, we do not need to compare again Bob's number with Alice in order to have shares C_{BA}^A and C_{BA}^B, where $C_{AB}^A + C_{AB}^B = 1$, since they are already provided by the third party in our Yao's protocol (see the Section 2.10). Thus, we will use it as the shares for the <Bob vs Alice> comparison.

1. *Alice (the first party) compares her value with all others, then sums up her parts of the shares and puts it as the first component in her shares vector. All other parties put their shares, that come from comparisons with Alice, again as the first component of their shares vector.*

[4]Here the subscript AB means <Alice vs Bob> comparison, whereas the subscript BA means <Bob vs Alice> comparison.

2. *Bob (the second party) compares his value with all others*[5]*, then sums up his parts of the shares and puts it as the second component in his shares vector. All other parties put their shares, that come from these comparisons as the second component of their shares vector.*

3. *The protocol continues until each party's value will be compared with all others.*

This provides the information shown in the Table 3.1 where C^i_{ij} *belongs to* P_i,

$\mathbf{P_1}$	$\mathbf{P_2}$	$\cdots$	$\mathbf{P_m}$
$\sum_{j=2,..,m} C^1_{1j}$	C^2_{12}	$\cdots$	C^m_{1m}
C^1_{21}	$\sum_{j=1,..,m}^{j\neq 2} C^2_{2j}$	$\cdots$	C^m_{2m}
$\vdots$	$\vdots$	$\ddots$	$\vdots$
C^1_{m1}	C^2_{m2}	$\vdots$	$\sum_{j=1,..,m-1} C^m_{mj}$

Table 3.1: Shares after all comparisons.

C^j_{ij} *belongs to* P_j*, and*

$$C^i_{ij} + C^j_{ij} = \begin{cases} 1 & \text{if } v_i > v_j, \\ 0 & \text{if } v_i \leq v_j. \end{cases}$$

Note that now, each column is owned by one party only; therefore we can treat them as the separate vectors distributed to each party. Moreover, for each party P_i*, the sum of the elements in the i-th row is* $\sum_{j=1,..,m}^{j\neq i} C^i_{ij} + \sum_{j=1,..,m}^{j\neq i} C^j_{ij}$*, which will show us exactly how many* v_j *were smaller than* v_i*. The problem now reduces to finding the ID of the maximum value in a sum of vectors*[6]*. This can be performed in SMC with* Maximum Value in the Sum of Vectors *protocol [8] where*

[5]Note that Bob does not need to compare with Alice again. He rather uses the other share provided from the third party.

[6]If all distances are different we know the maximum value is always $m-1$. Our protocol works even if some comparisons are between equal values. The use of circuit evaluation for a Yao comparison with shares faces the additional complexity of handling the case when the millionaires hold equal values. We believe solutions that resort to adding a second key (like the index of the vector [87]) to split up ties lead to larger circuits and more impractical solutions.

no party learns anything except the id of the entry holding the maximum value. If a version with shares is needed, the party P holding the maximum value M can generate a random number $s_P = R$, so that $s = M - R$ is made public to another party. The two parties then will hold values so that $s_p + s = M \mod F$.

It is not hard to see that this algorithm is as secure as our comparison protocol, because parties do not learn winners or losers of the comparisons since they are all encoded in distributed shares.

Using our Yao-comparison with shares and our Protocol 9 increases the complexity of the Chessboard distance to quadratic in the number of parties. However, the number of parties would be of the order of about 10 and thus, very affordable for the additional privacy.

3.2.3 Combinations of Metrics

Protocol 9 for the Chessboard distance and Protocol 8 with shares illustrated with the *Minkowski* metric are powerful enough to handle the fact that, in k-NN queries among parties sharing vertically partitioned data, it is likely the attributes may belong to very diverse domains. Each party may be applying a local metric M_{P_i} to the projection the party holds of the two records $\vec{p}$ and $\vec{q}$. This results in the value $v_i = M_{P_i}(\vec{p}, \vec{q})$. Thus, the global metric could be a weighted sum $\sum_{i=1}^{m} \omega_i v_i$ of the local metric values v_i. The problem of computing it would be solved by the Add vectors protocol as we illustrated with the *Minkowski* distance (with both versions, with shares or disclosing the global metric value). Alternatively, the global metric could be a weighted maximum $\max_{i=1}^{m} \omega_i v_i$. In this case, our Protocol 9 (previously illustrated with the Chessboard metric) generalizes to compute the global metric from the local metric values on the attributes known to each corresponding party. This allows for very flexible metrics that take into account issues like different units of measure and data types on the attributes.

3.2.4 Other Metrics

Other metrics common for large records (for example, between Web visitation paths [46]) are very important for high dimensional settings. The first one of these metrics is the Hamming distance H. Here $H(\vec{p}, \vec{q})$ is the number of entries where the vectors $\vec{p}$ and $\vec{q}$ differ. One realizes that for vertically partitioned data, secure multi-party computation of this reduces to computing again the sum of the Hamming distance in the projection by each party. If we now consider metrics, like the usage/access metrics or frequency metrics, we see that these metrics have the form $\vec{p}^T \cdot \vec{q}/\|\vec{p}\|_2\|\vec{q}\|_2$. That is, they are the cosine of the angle between the vectors $\vec{p}$ and $\vec{q}$. The scalar product $\vec{p}^T \cdot \vec{q}$ is again, the sum of values that correspond to the scalar product in the local projection of each party and we have already indicated how to perform Euclidean distances like $\|\vec{p}\|_2$. However, while the value

$$\cos(\alpha) = \frac{\vec{p}^T \cdot \vec{q}}{\|\vec{p}\|_2\|\vec{q}\|_2}$$

can be computed securely by our earlier protocols, we now show that it can also be computed securely and split in shares $s_a + s_b = \cos(\alpha)$ where again, s_a is known by Alice only and s_b is known by Bob only. This will pave the way for using this metric in our associative query algorithms later. Since dot products on vertically partitioned data are sums and can be computed with shares by Protocol 8 we have constants A_1, A_2 and A_3 known only by Alice and B_1, B_2 and B_3 only known by Bob so that

$$\begin{aligned}
\cos(\alpha) &= \frac{A_1 + B_1}{(A_2 + B_2)(A_3 + B_3)} \\
&= \frac{A_1 + B_1}{A_2A_3 + A_2B_3 + B_2A_3 + B_2B_3} \\
&= \frac{A_1 + B_1}{A_2A_3 + (A_2, A_3)^T \cdot \begin{pmatrix} B_3 \\ B_2 \end{pmatrix} + B_2B_3} \\
&= \frac{A_1 + B_1}{\underline{A_2A_3} + \underline{A_4} + \underline{\underline{B_4}} + \underline{\underline{B_2B_3}}} = \frac{A_1 + B_1}{A_5 + B_5},
\end{aligned}$$

where A_4 and A_5 are known only by Alice and B_4 and B_5 are known only by Bob. Note that $(A_2, A_3)^T \cdot \begin{pmatrix} B_3 \\ B_2 \end{pmatrix}$ is computed by the SCALAR PRODUCT PROTOCOL

WITH PRIVATE SHARES (see Section 2.7), which provided secret shares A_4 and B_4.

This last division could be computed securely by applying a SECURE DIVISION PROTOCOL [36] if we need output without shares, but this does not result in a metric split on private shares useful for k-NN queries. Alternatively, we can apply the SECURE DIVISION PROTOCOL (see Section 2.11) which provides an output with private shares.

3.3 Privacy-preserving k-Near Neighbours

3.3.1 Horizontally Partitioned Data

This section describes our privacy-preserving k-NN algorithm. Later, we will show that with this operation we can build a privacy preserving *SASH*.

For the PP-k-NN protocol we are given a set of vectors

$$\vec{v}_1^T = (v_{11}, \cdots, v_{1n}), \cdots, \vec{v}_m^T \quad = \quad (v_{m1}, \cdots, v_{mn}) \tag{3.1}$$

and a vector $\vec{q}^T = (q_1, q_2, \cdots, q_n)$, where m is the number of records/vectors involved in the computation. The goal is to find $NN(\vec{q}, k)$ where $NN(\vec{q}, k)$ is the set of indices of the k nearest neighbours to the vector $\vec{q}$. Assume the query vector $\vec{q}$ and the first $l < m$ vectors are owned by Alice while the other $m - l$ vectors are owned by Bob. In the case of the Euclidean distance, the distance values between $\vec{q}$ and $\vec{v}_i$, will be partially distributed between Alice and Bob. Alice will have

$$V^1_{q,v_{l+1}}, \cdots, V^1_{q,v_m} \tag{3.2}$$

and Bob will have

$$V^2_{q,v_{l+1}}, \cdots, V^2_{q,v_m}, \tag{3.3}$$

where $dist(\vec{q}, \vec{v}_i) = V^1_{q,v_i} + V^2_{q,v_i}, \quad l < i \leq m$. Of course, Alice will have also the distance values for her own data. At this stage, Alice and Bob can perform the ADD VECTOR PROTOCOL with the values that determine $dist(\vec{q}, \vec{v}_i) = V^1_{q,v_i} +$

$V^2_{q,v_i}, \quad l < i \leq m$. Alice receives these values shuffled by the permutation π that Bob knows. Alice finds among these values and her own $dist(\vec{q}, \vec{v_i}), 1 < i \leq l$, the k smallest. If any come from Bob's, she lets Bob know the indexes j and Bob returns $\pi^{-1}(j)$ to Alice. Then, Alice broadcasts the indexes of all k-NN. Note that Alice learns all the distances from $\vec{q}$ to data vectors of Bob included in k-NN. However, if only one vector comes from Bob, Alice will learn which exact vector has the given distance from the query vector $\vec{q}$, otherwise the indexes are shuffled, so for Alice it is not possible to know which vector has some particular distance. Moreover, even in the one vector case, when Alice learns the distance between Bob's vector and the query vector $\vec{q}$ it is not enough to obtain any private data (any coordinate) from Bob. Thus, the solution will still be secure in a weak security model (see Section 2.4). One should take into account that this solution is not very far from the ideal theoretical solution, because even in the ideal solution, parties are able to infer some information about the distribution of the vectors belonging to the other party. This issue is discussed more in Section 3.3.2.

Theorem 3 *The PP-k-NN protocol does not allow either Alice or Bob to learn each other's private data/vectors.*

Proof: In the first step of the PP-k-NN protocol, Alice obtains the values in List (3.2) and Bob obtains the values in List (3.3) as a result of the secure SCALAR PRODUCT PROTOCOL WITH PRIVATE SHARES (see Section 2.7). The next step applies the secure ADD VECTORS PROTOCOL (see Section 2.5) which allows Alice to learn the distance values. But, because the distances obtained by Alice were shuffled by Bob, Alice cannot learn the values in List (3.3). Clearly, Bob only learns the list of values in List (3.3) and the indexes of the k-NN. This, of course, is not enough to disclose Alice's private data. □

This protocol is efficient, because the secure SCALAR PRODUCT PROTOCOL WITH PRIVATE SHARES can be implemented very efficiently in linear time on the dimension n of the data. The secure SCALAR PRODUCT PROTOCOL WITH PRIVATE SHARES is executed at most m times for $O(mn)$ time. The secure ADD VECTORS PROTOCOL depends on homomorphic encryption. Because the ADD VECTORS PROTOCOL for PP-k-NN operates here with at most m values, the complexity of this phase only depends on m and not the dimension of the data. The total complexity is $O(mn)$ time, which is linear in the input as per List (3.1).

3.3.2 The Ideal Case For Privacy Preserving k-NN Queries

In this section we analyse what is the best possible security we can expect for a k-NN query. Assume Alice has a database $D_1 = \{\vec{a_1}, \cdots, \vec{a_m}\}$ and query vector $\vec{q}$, while Bob has database $D_2 = \{\vec{b_1}, \cdots, \vec{b_m}\}$. They want to compute privately

$$k\text{-NN}(\vec{q}, D_1, D_2) := \langle z_1, \cdots, z_k \rangle_{D_1 \bigcup D_2},$$

where $z_1, \cdots, z_k$ are the indexes of the vectors, which are k-NNs to the query vector $\vec{q}$.

As we mentioned earlier, the SMC literature has an ideal theoretical solution for this problem. If such an ideal (theoretically possible) protocol for k-NN were constructed from describing the function as a circuit, then every party obtains only the indexes $z_1, \cdots, z_k$. However, they also obtain what can be inferred from this. Assume $z_{i_1}, \cdots, z_{i_l}$ are indexes of vectors that belongs to Alice, and $z_{i_{l+1}}, \cdots, z_{i_k}$ are the those of vectors that belongs to Bob. If Bob constructs the $k-(l+1)^{th}$ order Voronoi diagram [24, 76] with his data, he can discover a cell/bounding box (BB) for the query vector $\vec{q}$. These cells are not regular, and in particular, the one for $\vec{q}$ could be extremely small even if the number $k-(l+1)$ is small. Naturally, the more of Bob's vectors among the k-NNs of $\vec{q}$, the more likely a more accurate cell/BB will be found. Thus, even in the ideal (theoretically possible) case, Bob is able to discover a cell/BB where the query vector $\vec{q}$ lies.

On the other hand, if Alice computes

$$d^a_{min} = \min_{a_i \in D_1 \text{ and } a_i \notin \{a_{z_{i_1}}, \cdots, a_{z_{i_l}}\}} (dist(\vec{q}, \vec{a_i})),$$

then Alice will know that all Bob's vectors included in the k-NN answer are in the hyper-sphere centred by $\vec{q}$ with radius d^a_{min}.

3.3.3 Application to Classification

Learning classifiers is a common machine learning or data mining task where from a training set of labelled cases, a classifiers is constructed to find the class of new instances. Famous approaches for this include Artificial Neural Networks, Decision Trees, Decision Lists and Support Vector Machines [100]. Recently, a solution for privately constructing k-NN classifiers in horizontally partitioned data has been proposed [67]. This solution is limited because it requires a non-colluding untrusted third party and it is computationally costly — $O(m^2k^2)$ where m is the number of parties.

Let us now see how we can use our PP k-NN protocol to solve this same problem far more efficiently and without a third party. We assume there are $m > 2$ parties and the query point $\vec{q}$ belongs to the first party. This is also an improvement from the previous solution that required the query point to be public (in our protocol, let the party that owns the query point be the first party). So, we are given $D_1, \cdots, D_m$ databases, where $D_i = \vec{p_1^i}, \cdots, \vec{p_n^i}$ are the attribute vectors for the i-th party. The t possible classes are identified by a set $L = l_1, \cdots, l_t$ of labels. Each point in $D_1 \cup \cdots \cup D_m$ has a label in L attached. The task is to find which of $l_j \in L$ corresponds to the query point $\vec{q}$ by taking a majority vote in the labels associated to the k-NN of $\vec{q}$. Now, using our PP k-NN protocol (see Section 3.3.1), the parties can compute the k-NN of the query vector $\vec{q}$. What we need now is to classify $\vec{q}$. The problem setting consists now of each party holding its training data vectors that are included in the global k-NN of the query point $\vec{q}$. Consequently, they know labels (classifiers) associated with their vectors. We want to compute which of the labels is the most present in the set of k-NN vectors. Assume the labels among the k-NN vectors are

$$L^{k\text{-NN}} = \langle (1)_{i_1}, \cdots, l^{(1)}_{i_{p_1}}, l^{(2)}_{i_1}, \cdots, l^{(2)}_{i_{p_2}}, \cdots, l^{(m)}_{i_1}, \cdots, l^{(m)}_{i_{p_m}} \rangle$$

where $p_1 + p_2 + \cdots + p_m = k$ and $l^{(j)}_{i_1}, \cdots, l^{(j)}_{i_{p_j}}$ are the labels of the vectors that the j-th party contributes to the k-NN. Then the label for $\vec{q}$ will be

$$l^q = Majority(L^{k\text{-NN}}). \tag{3.4}$$

Hence, the challenge is how to compute (3.4) privately.

In fact, this is equivalent to finding the row with the largest sum where each party owns a column, because $l_{i_1}^{(j)}, \cdots, l_{i_{p_j}}^{(j)}$ are the labels of the vectors that are known by the j-th party, he can compute sums of every label present in $l_{i_1}^{(j)}, \cdots, l_{i_{p_j}}^{(j)}$, if there are labels that were not present, just put 0. This is the column known by the j-th party. For example, assume the j-th party labels are $(l_1, l_2, l_2, l_1, l_4, l_4, l_1)$, then he will have $(3, 2, 0, 2, 0, \cdots, 0)$ as a representative column.

Hence, every party will know its representative vector with length t. The protocol must add them and find the index of the maximum value. This will represent the class label that will be assigned to $\vec{q}$. We now use our index-version of the finding MAXIMUM IN THE SUM OF VECTORS protocol (see Section 2.9) to add all representative vectors, and then Alice (the first party) can find the maximum value and the permuted index where the maximum lies. Because Bob was the author of permutation π, then Bob can apply π^{-1} to discover the original index where the maximum value lies. This will represent the label's index in L, so $\vec{q}$ will be labelled.

Example: Assume there are three parties and after computation of representative vectors they have $R^{(1)} = (2, 0, 0, 3, 2, 0)$, $R^{(2)} = (0, 1, 2, 0, 4, 0)$, $R^{(3)} = (0, 0, 1, 3, 1, 1)$. When we apply the secure add vectors protocol Alice gets a permutation of $R = (2, 1, 3, 6, 7, 1)$, so the id that represents the maximum value here is 5, then Bob needs to find the real id of the maximum and broadcast it. Thus, the label assigned for $\vec{q}$ will be $l_{\pi^{-1}(5)}$.

3.3.4 Vertically Partitioned Data

Given a set of vectors

$$\vec{v}_1 = (v_{11}, \cdots, v_{1m}), \cdots, \vec{v}_n = (v_{n1}, v_{n2}, \cdots, v_{nm})$$

together with a vector $\vec{q} = (q_1, q_2, \cdots, q_m)$ (where m is the number of parties involved in the computation), we must find $NN(\vec{q}, k)$ where $NN(\vec{q}, k)$ is the ids for the k nearest neighbours to the vector $\vec{q}$.

Protocol 10 *1. The parties engage in metrics calculation with shares. After this, we can assume the first party, Alice, holds a vector $\vec{s}^a$ of dimension n*

and Bob holds a vector $\vec{s^b}$ so that $s_i^a + s_i^b = dis(\vec{q}, \vec{v}_i)$ (in fact, it is possible to assume encryption modulo a field F; that is $s_i^a + s_i^b = dis(\vec{q}, \vec{v}_i) \mod F$).

2. *Now, Bob (the second party) uses a random value R_b only known to him, and adds the vector $\vec{R}_b = (R_b, ..., R_b)$ (that consists of all entries set to one random number R_b). He adds $\vec{s^b} + \vec{R}_b$.*

3. *Now Alice and Bob use the* ADD VECTORS PROTOCOL *for $\vec{s^a}$ belonging to Alice and $\vec{s^b} + \vec{R}_b$ belonging to Bob. This gives Alice a random translation of the distances $dist(\vec{q}, \vec{v}_i)$, for $i = 1, ..n$, permuted by a random permutation π that only Bob knows.*

4. *Alice sorts them and sends the top k ids to Bob, then Bob broadcasts $\pi^{-1}(IDs)$ (which are IDs of k-NN of $\vec{q}$) to all parties.*

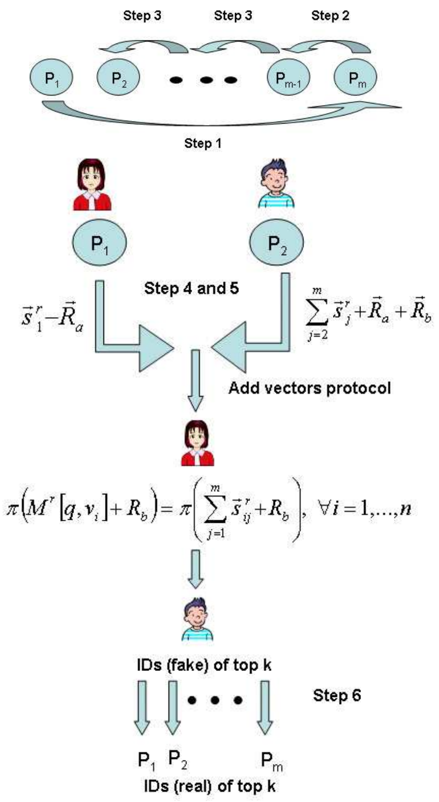

Step 3
Step 3
Step 2
P1
P2
Pm-1
Pm
Step 1
P1
P2
Step 4 and 5
$\vec{s}_1^{\,r} - \vec{R}_a$
$\sum_{j=2}^{m} \vec{s}_j^{\,r} + \vec{R}_a + \vec{R}_b$
Add vectors protocol
$\pi(M^r[q, v_i] + R_b) = \pi\left(\sum_{j=1}^{m} \vec{s}_{ij}^{\,r} + R_b\right), \forall i = 1, ..., n$
IDs (fake) of top k
Step 6
P1 P2 Pm
IDs (real) of top k

Figure 3.1: Algorithm for the calculation of k-NN.

Alice learns the distribution of the distance values translated by a random number. No other information is disclosed.

Theorem 4 *The PP k-NN protocol does not allow any party to learn another party's private data represented as an attribute in a feature vector.*

Proof: Clearly each party except Bob and Alice participate in the protocol with an input that looks random (under the theory of SMC, all these parties can be simulated by polynomial algorithms that use as inputs guessed values from an oracle). Bob also participates in the protocol with what can be random input until Alice passes to him π^{-1} of the IDs that constitute the result. This is secure because in the semi-honest model, Bob can learn anything that can be inferred from the result. Alice cannot be simulated with a polynomial algorithm that uses guessed values from an oracle or the protocol would fail. However, Alice cannot link any of the values it receives to any party (because it ignores π, none of the distance values becomes known because they have R_b added to them and only Bob knows R_b). $\square$

Note however, that, if the global metric is a sum of local metrics, one may be tempted to parallelise the Add vectors protocol and Alice would add one random vector $\vec{R}^a = (R_a, \ldots, R_a)$ to its projection of all the metrics, an pass it to the m-party, which would add its projection and pass the vector down to the $(m-1)$-th party, and so on. This does not represent a real saving except using one rather than n random values generated by Alice, but allows each party to learn the distribution of the projection of the distance values for the previous parties in the line-up.

Also note that if we use the versions with shares over a field for our protocols that compute metrics, then it is possible to use binary search over a field for the top k-neighbours [95]. The field binary search requires Yao's comparisons with shares (which we have now made more practical). Using field binary search would prevent Alice learning the distribution of the distance values at the expense of the $O(n \log |F|)$ Yao comparisons. Moreover, because the data is vertically partitioned, and the algorithm is distributed, it is impossible for one party to repeat an associative query many times without the participation (or knowledge) of the other parties. This is an extra security aspect of our context.

3.3.5 An Alternative Solution for k-NN

Given a set of vectors

$$\vec{v}_1 = (v_{11}, \cdots, v_{1m}), \cdots, \vec{v}_n = (v_{n1}, v_{n2}, \cdots, v_{nm})$$

together with a vector $\vec{q} = (q_1, q_2, \cdots, q_m)$ (where m is the number of parties involved in the computation), we must find $NN(\vec{q}, k)$ where $NN(\vec{q}, k)$ is the ids for the k nearest neighbours to the vector $\vec{q}$.

Note also that $dist(\vec{p}, \vec{q}) = d^a_{pq} + d^b_{pq}$ and $dist(\vec{p}, \vec{r}) = d^a_{pr} + d^b_{pr}$ (with d^a_{pr},d^a_{pq} known by Alice and d^b_{pq}, d^b_{pr} to Bob), the question whether "$dist(\vec{p}, \vec{q}) < dist(\vec{p}, \vec{r})$?" is the same as "$d^a_{pq} - d^a_{pr} < d^b_{pq} - d^b_{pr}$?".

Protocol 11 *1. The parties engage in metric calculation with shares. After this, we can assume the first party, Alice, holds a vector $\vec{d^a}$ of dimension n and Bob holds a vector $\vec{d^b}$ so that $d^a_i + d^b_i = dist(\vec{q}, \vec{v}_i)$ (in fact, it is possible to assume encryption modulo a field F; that is $d^a_i + d^b_i = dist(\vec{q}, \vec{v}_i) \bmod F$).*

2. Alice and Bob generate the following symmetric matrices locally:
Alice

$$D_A = \begin{pmatrix} 0 & |d^a_1 - d^a_2| & |d^a_1 - d^a_3| & \dots & |d^a_1 - d^a_n| \\ |d^a_2 - d^a_1| & 0 & |d^a_2 - d^a_3| & \dots & |d^a_2 - d^a_n| \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ |d^a_n - d^a_1| & |d^a_n - d^a_2| & |d^a_n - d^a_3| & \dots & 0 \end{pmatrix}$$

and Bob

$$D_B = \begin{pmatrix} 0 & |d^b_1 - d^b_2| & |d^b_1 - d^b_3| & \dots & |d^b_1 - d^b_n| \\ |d^b_2 - d^b_1| & 0 & |d^b_2 - d^b_3| & \dots & |d^b_2 - d^b_n| \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ |d^b_n - d^b_1| & |d^b_n - d^b_2| & |d^b_n - d^b_3| & \dots & 0 \end{pmatrix}.$$

3. Alice and Bob compare elements[7] of their matrices by Yao comparison with private shares which will provide them with the following matrices:

[7]Here, they need to compare only the upper diagonal part, because they are symmetric. However the matrix of shares after comparison will not be symmetric.

Alice obtains

$$Y_A = \begin{pmatrix} 0 & s^a_{12} & s^a_{13} & \dots & s^a_{1n} \\ s^a_{21} & 0 & s^a_{23} & \dots & s^a_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ s^a_{n1} & s^a_{n2} & s^a_{n3} & \dots & 0 \end{pmatrix}$$

and Bob obtains

$$Y_B = \begin{pmatrix} 0 & s^b_{12} & s^b_{13} & \dots & s^b_{1n} \\ s^b_{21} & 0 & s^b_{23} & \dots & s^b_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ s^b_{n1} & s^b_{n2} & s^b_{n3} & \dots & 0 \end{pmatrix}.$$

4. *They compute the sum of the elements in each row and generate the following two vectors. Alice obtains*

$$VA = \begin{pmatrix} \sum_{j=1}^n s^a_{1j} \\ \sum_{j=1}^n s^a_{2j} \\ \vdots \\ \sum_{j=1}^n s^a_{nj} \end{pmatrix}$$

and Bob obtains

$$VB = \begin{pmatrix} \sum_{j=1}^n s^b_{1j} \\ \sum_{j=1}^n s^b_{2j} \\ \vdots \\ \sum_{j=1}^n s^b_{nj} \end{pmatrix}.$$

5. *The only thing left is to apply the* SECURE ADD VECTORS PROTOCOL, *which will allow Alice to sort the numbers* [8] *in increasing order and send the ids of the first k to Bob, so that Bob can apply π^{-1}, to recover the true indices, and broadcast them to other parties.*

Note that, Alice only learns counts of $d^a_{pq} - d^a_{pr} < d^b_{pq} - d^b_{pr}$ comparisons. She does not learn the distribution of the distance values and no other information is disclosed. The security proof follows from the composition theorem. Clearly, the price to pay for the added security of this protocol is that it is quadratic on n.

[8]Note that this time it is not the actual distances, but the numbers counting how many vectors are before the from query vector than the current vector; that is, the ranking.

Chapter 4

Privacy Preserving Data Structures

In this chapter we will present privacy-preserving data structures. As we mentioned earlier, in order to be efficient while performing associative queries we need data structures that support these kinds of queries. Hence, the need for privacy-preserving data structures for big databases is not avoidable. We also focus on k-NN queries in the section describing our experiments. Note also that k-NN is the most used query in information retrieval. From our experiments one can see the power of having privacy-preserving data structures in order to perform privacy-preserving k-NN queries. This is disscussed again in the section describing our experiments (see Section 4.3.4). There we compare k-NN computations by Fagins A0 algorithm, which does not operate on sophisticated data structures, with the k-NN computations with the help of the *SASH* data structure.

In order to be consistent, as much as possible we use the original notation in the discussion of KD-Trees. This results in some differences with respect to our notation, but they are not significant. In the origianl literature of KD-Trees K is used for the number of dimensions [1]. We will use K as the number of dimensions and keep k for the number of near neigbours in a k-NN query.

[1]Recall that we assume the existence of virtual parties that ensures equality of dimensions and the number of parties.

4.1 Privacy-Preserving R-Trees for Vertically Partitioned Data

An R-Tree [56] is a height-balanced tree (all leaves appear on the same level) similar to a B-tree. Although R-Trees are designed for spatial objects with multidimensional keys stored in the leaf nodes, here the R-Tree is used with objects that are vectors identified as points in multi-dimensional real space. Nodes correspond to disk pages if the index is disk-resident, and the structure is designed so that an associative query requires visiting only a small number of nodes. The structure is completely dynamic, allowing insertions and deletions that can be intermixed with searches, and it requires no periodic reorganisation.

Leaf nodes in an R-Tree contain entries of the form $(I, \vec{p})$ where $\vec{p}$ refers to the point (or key) in the database and I is a K-dimensional bounding box for $\vec{p}$ that has the form $I = (I_0 \times I_1 \times \ldots \times I_{K-1})$. Thus, for each index record $(I, \vec{p})$ in a leaf node, I is the smallest rectangle that spatially contains the K-dimensional data object represented by the indicated key. Thus, I can be identified with $\vec{p}$ on the leaves.

Internal nodes contain entries of the form $(I, \mathit{child\_pointer})$ where $\mathit{child\_pointer}$ is the address of a lower node in the R-Tree and $I = I_0 \times I_1 \times \ldots \times I_{K-1}$ is again a K-dimensional box. The intervals I_i are closed bounded intervals describing an extent on one domain, in particular, an extent in the i-th domain. Most importantly, the box contains (sometimes called "covers" or "includes") all boxes in all the lower node's entries for the subtree rooted at (pointed by) $\mathit{child\_pointer}$. That is, for each entry $(I, \mathit{child\_pointer})$ in a non-leaf node, I is the smallest rectangle that spatially contains the rectangles in all its descendant nodes. Under vertically partitioned data, every I_i is securely stored by one and only one of the parties involved.

Another property of R-Trees is their utilisation of the storage on a page of the index. Let M be the maximum number of entries that will fit in one node and let $m \leq \frac{M}{2}$ [2] be a parameter specifying the minimum number of entries in a node.

[2]The original R-Tree notation uses m in this form, but in this thesis, it has been used before as the number of records in horizontally partitioned data, or the number of parties.

Thus, every leaf node contains between m and M index records unless it is the root. Every non-leaf node has between m and M children unless it is the root. The root node has at least two children unless it is a leaf. Figure 4.1 shows the structure of a two-dimensional (2D) R-Tree and illustrates the containment and overlapping relationship that can exist between rectangles.

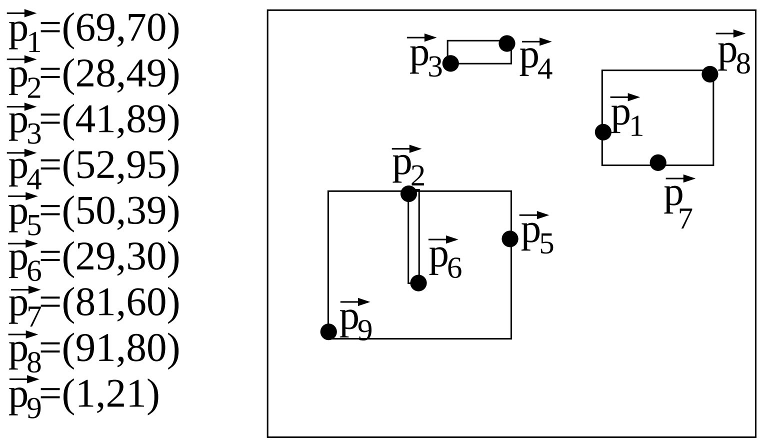


Figure 4.1: Example of 2D R-Tree for 9 points.

The entire structure (not actual private data) of the privacy-preserving R-Tree is known to all parties. However, in leaves of the form $(I, \vec{p})$, because $\vec{p}$ is essentially I, each party will only know the attributes in the domain it controls. In internal nodes $(I, child\_pointer)$, each party will know those closed intervals I_i in the domain of its control for each $I = I_0 \times I_1 \times \ldots \times I_{K-1}$. A split data structure has been used before in the literature on privacy-preserving classification [39].

For illustration, we use two parties in Figure 4.2. It shows two mirror R-Trees that exist for Alice and Bob. Each of them only knows his/her domain intervals in internal nodes while both know the structure of the R-Tree (that is, which nodes are pointed to which, which is the root and where all pointers points to). They also know the identifier of those attribute-value vectors at the leaves of the R-Tree. What they do not share is the boxes I on the internal nodes. The figure uses a simple notation on bounding boxes for the domain of the party that controls that domain (see Figure 4.1). Because we are dealing in 2D, Figure 4.2 uses bounding boxes of the form $BB = A_i^l \times B_r^s$, where $A_i^l = [i, l]$ is an interval known by Alice and only Alice, and $B_r^s = [r, s]$ is an interval known by Bob and only Bob. For each of the points $\vec{p_i} = (a_i, b_i)$, Alice and only Alice knows a_i and Bob and only Bob knows b_i. To operate a privacy-preserving R-Tree all parties

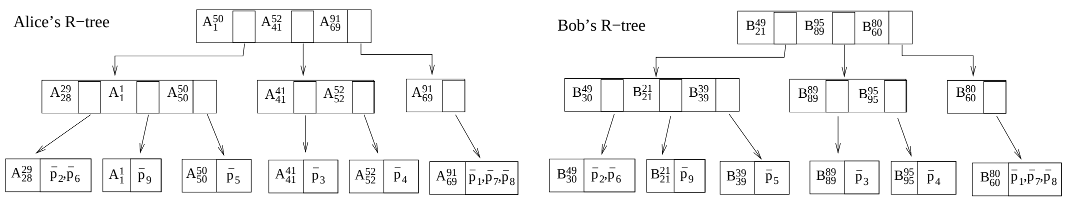

Alice's R−tree
A_1^{50} A_{41}^{52} A_{69}^{91}
A_{28}^{29} A_1^1 A_{50}^{50}
A_{41}^{41} A_{52}^{52}
A_{69}^{91}
A_{28}^{29} $\bar{p}_2,\bar{p}_6$
A_1^1 $\bar{p}_9$
A_{50}^{50} $\bar{p}_5$
A_{41}^{41} $\bar{p}_3$
A_{52}^{52} $\bar{p}_4$
A_{69}^{91} $\bar{p}_1,\bar{p}_7,\bar{p}_8$
Bob's R−tree
B_{21}^{49} B_{89}^{95} B_{60}^{80}
B_{30}^{49} B_{21}^{21} B_{39}^{39}
B_{89}^{89} B_{95}^{95}
B_{60}^{80}
B_{30}^{49} $\bar{p}_2,\bar{p}_6$
B_{21}^{21} $\bar{p}_9$
B_{39}^{39} $\bar{p}_5$
B_{89}^{89} $\bar{p}_3$
B_{95}^{95} $\bar{p}_4$
B_{60}^{80} $\bar{p}_1,\bar{p}_7,\bar{p}_8$

Figure 4.2: Example of shared R-Tree structure, but different bounding boxes per domain, where Alice knows the first coordinate and Bob the second.

will become aware of the outcomes of comparisons, but no party will learn the values of the other parties. All parties are included at each level of the tree. We explain our methods with two parties not only to get the main ideas across, but because this is usually the difficult case. When the details differ for more parties, we will make a note of it.

The fundamental operations for a privacy-preserving R-Tree are insertions, deletions and two types of searches: a search to find if a point $\vec{p}$ is in the database and the fundamental associative query to retrieve all points in an ϵ-neighbourhood. In R-Trees, these operations are built upon other operations. The road-map of these dependencies and how we will implement classic building blocks for R-Tree operations with SMC-operations is shown in Figure 4.3, where PP stands for Privacy-Preserving variant. As with B-trees, in order to keep the tree balanced, nodes are split. This happens when a node reaches its capacity M (an attempt to hold $M + 1$ entries $(I, \mathit{child\_pointer})$ on a node results in a split). When a node is split, two new bounding boxes need to be selected, and the $M+1$ boxes of the original node must be distributed among the new boxes. Thus, the algorithm for the operation to split a node relays in the operation PickSeeds (in order to select two new boxes) and PickNext (in order to distribute children of the split node into the new boxes). We start with the splitting and then go into its two sub-operations.

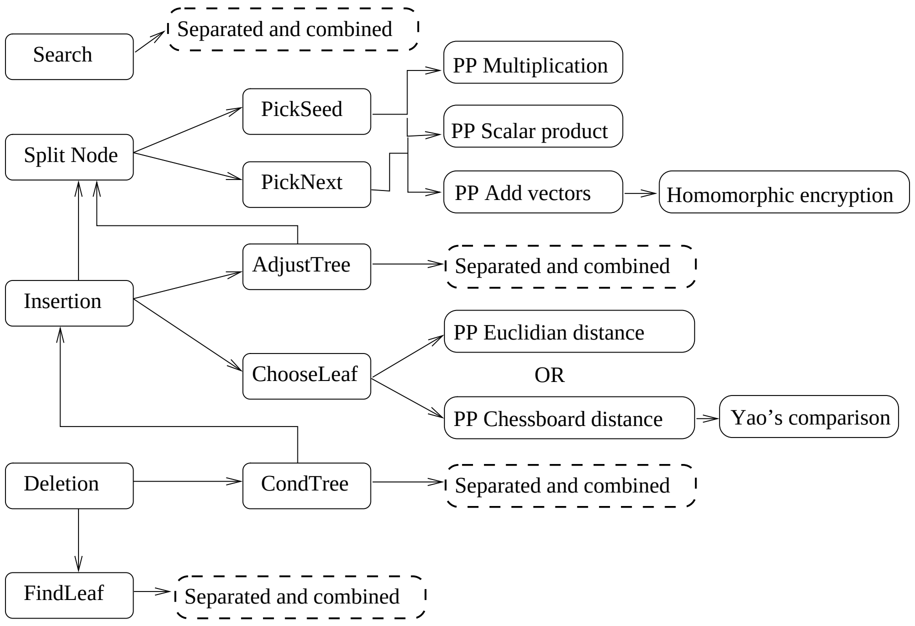


Figure 4.3: Algorithms as building blocks for other algorithms.

4.1.1 Splitting the Rectangles or Algorithm SplitNode.

In the literature, there are many ways to split boxes in R-Trees from the root to leaf nodes and each way has its advantages and disadvantages. Here, we are more interested in splitting while preserving privacy of the parties involved. Thus, we focus on the classical splitting algorithm [56] with quadratic cost.

This algorithm attempts to find a small-volume split, but is not guaranteed to find one with the smallest volume possible. The computational cost is quadratic in M and linear in the number K of dimensions. The algorithm picks two of the $M+1$ entries to be the first elements of the two new groups, by choosing the pair that would take up the most area; that is, the box covering both entries would be greatest. The remaining entries are then assigned to groups one at a time. At each step, the area expansion required to add each remaining entry to each group is calculated, and the entry assigned is the one showing the greatest difference between the two groups. This informal description is expressed in specific steps to divide the set of $M+1$ index entries into two groups as follows.

1. Apply Algorithm PickSeeds to choose two entries to be the first elements

of the groups. Assign each of these two entries to a group.

2. If all entries have been assigned, stop. If one group has so few entries that all the rest must be assigned to it in order for it to have the minimum number m, assign them and stop.

3. Invoke Algorithm PickNext to choose the next entry to assign. Add it to the group whose covering rectangle will have to be enlarged least to accommodate it. Resolve ties by adding the entry to the group with smaller area, then to the one with fewer entries, then to either. Repeat from Step 2.

Naturally, the parties can apply this algorithm preserving privacy provided that they can apply PickSeeds and PickNext preserving privacy. We will look at these two building blocks now.

Algorithm PickSeeds**.**

Select two entries to be the first respective element of two new groups.

1. For each pair of entries E_1 and E_2, compose a bounding box J for E_1 and E_2. Calculate $V = Vol(J)$, the volume of J.

2. Choose the pair with the largest V.

Now in order to achieve privacy here we need to know how to calculate $Vol(J)$. Due to security reasons we will separate the two party case from the multiparty ($K \geq 3$) case. Note, that even if one party owns more than one attribute, computing the volume is the product of the volume values on the projections. First let us see how we can calculate the volume when $K \geq 3$ (three or more parties are involved). It is not hard to reduce computing the bounding box of two entries to computing the bounding box of two extreme points (each party just needs to find the extreme values under the domain it controls for all the points involved in the entries). Let us assume we are given two extreme points

$\vec{p}_1 = (v_1^x, v_2^x, \cdots, v_K^x)$ and $\vec{p}_2 = (v_1^y, v_2^y, \cdots, v_K^y)$. The goal is to calculate the K-dimensional volume which includes these two points (when each party controls one and only one of the dimension). The volume is

$$|v_1^x - v_1^y| \times |v_2^x - v_2^y| \times \cdots \times |v_K^x - v_K^y|.$$

This multiplication can be performed by the following algorithm.

Protocol 12 *(Secure multiplication algorithm)*

- *(VC1) The first party produces a random number r and sends* $\log(r)$ *to the second party.*
- *(VC2) The second party adds* $\log(r|v_1^x - v_1^y|) + \log(|v_2^x - v_2^y|)$ *and sends it to the third party.*
- *(VC3) Continue until K^{th} adds the sum with* $\log(|v_K^x - v_K^y|)$ *and sends it to the first party.*
- *(VC4)Then the first party adds* $\log(|v_1^x - v_1^y|)$*, computes the exponential of the obtained sum and divides by r.*

Under the semi-trusted model when parties do not collude they do not learn each others' data. Note that this is the SECURE SUM PROTOCOL of the logarithms.

Since we have to calculate for all possible pairs E_i and E_j and then take the one with largest V (see PEEKSEEDS algorithm Step 1), the only thing left is to calculate the largest value of the obtained K-dimensional volumes. This can be performed securely as the first party (who already has all volumes possible) can calculate the largest of the volumes and send it to the other parties, so all parties learn which pair (E_i and E_j) produces the largest volume. Obviously if we have more than one pair then Alice learns all volumes of all possible pairs which is not so desirable. In this case we can use a version of the above protocol in secret shares, namely if we halt the protocol in step (VC3), then K^{th} party will have

$$\sum_{i=2..K} \log(r|v_i^x - v_i^y|)$$

and the first party will have $\log\left(\frac{1}{r}|v_1^x - v_1^y|\right)$, where the exponential of the sum will give the volume sought. Hence, the first party and the K^{th} party will hold values of all possible volumes in secret shares. It only remains to apply Maximum value in the sum of the vectors protocol, with an extra exponent computation from the first party side, to find the maximum value.

To illustrate this let us see how it works in a two party case. Note that we must use a version with shares only in this case, otherwise it immediately reveals the private value of the second party. Since, $|v_1^x - v_1^y| \times |v_2^x - v_2^y| = \exp(V_1 + V_2)$, where $V_1 = \log(|v_1^x - v_1^y|)$ is known by Alice and $V_2 = \log|v_2^x - v_2^y|$ is known by Bob.

Since there are $M+1$ bounding boxes for the entries being considered in the splitting of the node, there are $\left(\begin{array}{c} M+1 \\ 2 \end{array}\right)$ pairs. The task is to find

$$\max_{\forall i,j}(vol(Bounding\_Box(E_i, E_j)))$$

where i, j denotes the indices to identify one of the possible pairs of boxes. Note that

$$\begin{aligned} & Vol(Bounding\_Box(E_i, E_j)) \\ &= Vol(Pr_A(Bounding\_Box(E_i, E_j))) \times \\ &\quad Vol(Pr_B(Bounding\_Box(E_i, E_j))) \\ &= \exp(V_A + V_B), \end{aligned} \tag{4.1}$$

where Pr_A and Pr_B are the projections only known by Alice and Bob respectively.

Thus, for two parties we have translated the question of the largest value of distributed products to the largest value of distributed sums. Thus, the question becomes how to calculate

$$\max_{l=l(i,j)} \exp(V_A(l) + V_B(l)). \tag{4.2}$$

Here we can use again Maximum value in the sum of the vectors protocol which uses the add vectors protocol (see Section 2.5). Thus, Alice will obtain $\pi(V_A + V_B)$ and then find the largest value of the entries. Say l_0 is the index where

$$\pi \exp(V_A(l_0) + V_B(l_0)) = \max_l (\exp \pi(V_A(l) + V_B(l))) .$$

Alice sends l_0 to Bob, so now they know which pair E_i and E_j gives the largest joint area. Hence, neither Alice nor Bob learns the values of their data records but they learn which pair of records has the largest area.

Algorithm PickNext.

Select one remaining entry for classification in a group.

1. For each entry E not yet in a group, calculate d_1 = the volume increase required in the covering box of Group 1 to include E. Calculate d_2 similarly for Group 2.
2. Choose any entry with the maximum difference between d_1 and d_2.

The question now is how to calculate securely the increased volume when we want to include one additional point or entry into an already existing box? (see Figure 4.4). Thus, the problem is that given the box as shown in the Figure 4.4 with the corner points $\vec{p}_1$ and $\vec{p}_2$, we need to calculate the volume that will be added when we increase the box so that it will include a new point $\vec{p}_3$ as well. In the K-dimensional case, rather than area, we will be dealing with the K-dimensional volume. It is clear that, in K dimensions, the increase in volume ΔVol can be calculated in the following way.

$\Delta Vol = Vol(\text{New box}) - Vol(\text{Old box}) = [v_1 + V_1] \times \cdots \times [v_K + V_K] - [v_1] \times \cdots \times [v_K]$,

where $v_i = |v_i^x - v_i^y|$ is the projection of the K-dimensional volume in the i^{th} direction, V_i is the increased projection, again in the i^{th} domain. For example, let $\vec{p}_1 = (s_1^1, \cdots)$, $\vec{p}_2 = (s_1^2, \cdots)$ and $\vec{p}_3 = (s_1^3, \cdots)$, then $v_1 = |s_1^2 - s_1^1|$. Now,

- if s_1^3 is less than $\min(|s_1^1|, |s_1^2|)|$, then $V_1 = |s_1^3 - \min(|s_1^1|, |s_1^2|)|$;
- if s_1^3 is larger than $\max(|s_1^1|, |s_1^2|)|$, then $V_1 = |s_1^3 - \max(|s_1^1|, |s_1^2|)|$;

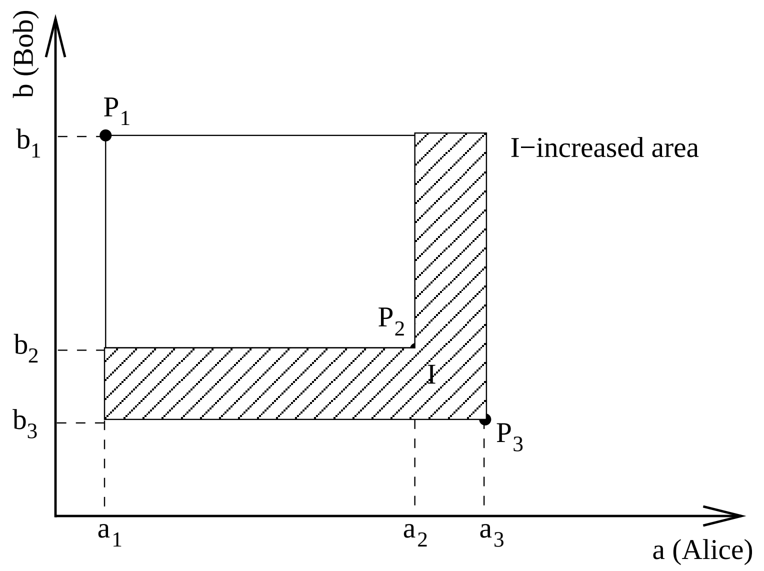


Figure 4.4: How the increased area is calculated

- otherwise $V_1 = 0$.

So, the only thing left to do now is to complete this step is to apply the Secure multiplication algorithm we used for calculating $Vol(J)$ in the PickSeeds algorithm. However, as in the algorithm PickSeeds, we can take advantage of the fact that the computation of which entry is to be assigned to Group 1 and Group 2 is to be performed for $M+1-2$ entries first. Then, for $M+1-3$ entries (because the bounding box of at least one of the groups changes every time an entry is assigned to a group). However, Alice and Bob never perform this exercise for less than m entries (thus, they are always operating with permuted vectors). That is, in the case of two parties, Alice and Bob compute simultaneously for all entries E_i (not assigned to a group) the values $d_1^i = \Delta Vol(Group1(E_i))$ and $d_2^i = \Delta Vol(Group2(E_i))$ (see Step 1 in PickNext). As a result they obtain the index of the entry for which $|d_2^i - d_1^i|$ is largest.

Now we move to the searching algorithm.

4.1.2 Algorithm Search.

The search algorithm descends the tree from the root in a manner similar to B-tree. However, more than one subtree under a node visited may need to be searched, hence, as it is standard with R-Trees, it is not possible to guarantee good worst-case performance. In the following, we denote the box part of an index entry E by EI, and the *child_pointer* part by Ep. The associative queries is, given a simple shape S (like a box or a ϵ-neighbourhood) and an R-Tree whose root node is T, find all points contained by the search shape S. The simple algorithm for this query is as follows.

1. If T is not a leaf, check each entry E to determine whether EI overlaps S. For all overlapping entries, invoke Search on the tree whose root node is pointed to by Ep.

2. If T is a leaf, check all entries E to determine whether EI overlaps S. If so, E is the qualifying point.

In the searching algorithm we have to find all boxes EI which overlap the given shape S. When S is a box, we only have to check projections of EI with projections of S. An illustration with two parties shows how this is achieved. Since $EI = [a_1, a_2] \times [b_1, b_2]$, Alice checks whether $S_a = Pr_A(S)$ overlaps $[a_1, a_2]$ and Bob checks whether $S_b = Pr_B(S)$ overlaps $[b_1, b_2]$, where $Pr_A(S)$ is the projection on Alice's domain and $Pr_B(S)$ is the projection on Bob's domain). In the case of more than two parties, every party checks appropriate projections ($Proj_i(EI)$ and $Pr_i(S)$, for the i^{th} party) on his/her domain. All parties must report an intersection in their domain for box S to overlap EI. If any party misses an overlap, then S and EI do not overlap.

The case for an ϵ-neighbourhood is slightly more delicate and also typically a slightly different algorithm is used. In such an algorithm, the point $\vec{p}$ that is the centre of the neighbourhood is found first (using standard search with $S = \vec{p}$). Then, the algorithm proceeds from the leaf where $\vec{p}$ was found up the tree invoking the Search algorithm with $S = \epsilon$-neighbourhood on all nodes not already searched. In this case, the overlap at each non-leaf involves testing if

$[\epsilon - Pr_i(\vec{p}), Pr_i(\vec{p}) + \epsilon]$ intersects the projection on the i-th domain; while for leaves, a secure distance computation is required.

We also note that the case where the shape S is a point $\vec{p}$ essentially works as we described before, treating S as a box with the upper limit equal to the lower limit in all projections. However, when we expect to find $\vec{p}$ exactly once, the algorithm is called FindLeaf. We do not feel it is necessary to reproduce further details for this more specific search algorithm. Moreover, in clustering it may be possible that a point $\vec{p}$ appears more than once; but, the name FindLeaf is used by the deletion algorithm. Before discussing deletions, we proceed with insertions.

4.1.3 Algorithm for Insertion.

The task here is to insert a new index entry E into an R-Tree. It is not possible to invoke FindLeaf because we may not find the point (that is probably the main reason it is being inserted). Although FindLeaf may find a leaf where it can be inserted if not found, it may cause the enlargement of the overlap between sibling nodes up the tree. Keeping the overlap to a minimum is crucial to ensure good performance on searches (recall that the depth of a tree is the logarithm of its size with the value of the base the parity of nodes). Searches go down siblings that overlap causing extra work besides a path from the root to a leaf. Heuristically, overlap during insertion is minimised by the sub-algorithm ChooseLeaf, while the possible overload of a node when inserting a new value is handled by SplitNode (which we discussed already). The propagation of splits up the tree is the responsibility of AdjustTree. These pieces compose the insertion algorithm below.

1. Invoke ChooseLeaf to select a leaf node L in which to place E.

2. If L has room for another entry, install E. Otherwise invoke SplitNode to obtain L and LL containing E and all the old entries of L.

3. Invoke AdjustTree on L, also passing LL if a split has been performed.

4. If node split propagation caused the root to split, create a new root whose children are the two resulting nodes.

Algorithm ChooseLeaf.

The task is to select a leaf in which to place a new entry E.

1. Set N to be the root node.
2. If N is a leaf, return N.
3. If N is not a leaf, let F be the entry in N whose box FI needs least enlargement to include EI. Resolve ties by choosing the entry with the box of smallest volume.
4. Set N to be the child node pointed to by Fp and repeat from step 2.

The only step that applies operations on private data by parties is the third step. However, the data inserted here is a point $\vec{p}$ (and not a spatial object with non-zero volume). The parties can compute the enlargement it would produce for all entries on N. We have already shown how to achieve this securely for 3 or more parties (and this was illustrated with the insertion of point $\vec{p}_3$ in Figure 4.4).

Algorithm AdjustTree.

The task here is to ascend from a leaf node L to the root, adjusting covering boxes and propagating node splits as necessary.

1. Set $N = L$ if L was split previously, set NN to be the resulting second node.
2. If N is the root, stop.
3. Let P be the parent node of N, and let $E_N I$ be so that it tightly encloses all entry rectangles in N.

4. If N has a partner NN resulting from an earlier split, create a new entry $E_{NN}I$ with $E_{NN}p$ pointing to NN and $E_{NN}I$ enclosing all rectangles in NN. Add E_{NN} to P if there is room. Otherwise, invoke SplitNode to produce P and PP containing E_{NN} and all P's old entries.

5. Set $N = P$ and set $NN = PP$ if a split occurs. Repeat from step 2.

Recall that the structure of the tree is shared by all parties, so the only privacy concerns appear in Step 3 and Step 4. The main issue is how to compute securely the box that tightly encloses all of a given a number of boxes. We have already argued that finding this corresponds to finding extremes in each domain. For example, with two parties, given a set of n boxes

$$r_1 = [a_{11}, a_{12}] \times [b_{11}, b_{12}], \cdots, r_n = [a_{n1}, a_{n2}] \times [b_{11}, b_{12}]$$

we must find $R = [A_1, A_2] \times [B_1, B_2]$ such that $\bigcup_{i=1..n} r_i \subset R$ and no smaller box R has this property. Let $A_1 = \min(a_{11}, ...a_{n1})$, $A_2 = \max(a_{12}, ...a_{n2})$, $B_1 = \min(b_{11}, ...b_{n1})$, and $B_2 = \max(b_{12}, ...b_{n2})$. This is clearly a solution and each max and min operation here can be performed by each party in isolation (the best assurance possible that privacy is preserved).

We are now ready for deletions.

4.1.4 Algorithm Delete.

Deletions may cause the reverse operation of a node split. This merging of nodes is handled by CondenseTree. To remove the index record E from an R-Tree we follow these steps.

1. Invoke FindLeaf to locate the leaf node L containing E. Stop if the record is not found.

2. Remove E from L.

3. Invoke CondenseTree, passing L.

4. If the root node has only one child after the tree has been adjusted, make the child the new root.

Algorithm Condense**T**ree.

Given a leaf node L from which an entry has been deleted, eliminate the node if it has too few entries and relocate its entries. Propagate node elimination upward as necessary. Adjust all covering rectangles on the path to the root, making them smaller if possible.

1. Set $N = L$. Set Q, the set of eliminated nodes, to be empty.
2. If N is the root, go to step 6. Otherwise let P be the parent of N, and let E_N be N's entry in P.
3. If N has fewer than m entries, delete E_N from P and add N to set Q.
4. If N has not been eliminated, adjust $E_N I$ to tightly contain all entries in N.
5. Set $N = P$ and repeat from step 2.
6. Re-insert all entries of nodes in set Q. Entries from eliminated leaf nodes are reinserted in tree leaves as described in Algorithm Insert, but entries from higher-level nodes must be placed higher in the tree, so that leaves of their dependent subtrees will be on the same level as leaves of the main tree.

All steps are tree structure steps, except Step 4. However, we again face the situation where we need to find a tight box for a set of entries (and we have demonstrated this can be done without data exchange, Insert which is already secure. Thus, deletions can preserve privacy as well.

4.1.5 Analysis

We use privacy-preserving R-Trees to perform privacy-preserving $DBSCAN$ and this cannot be obtained without a cost. We will first analyse the overhead on the time resources (i.e. the additional operations relative to the naive alternative of concentrating the data with only one party who then performs the clustering). Observe first that SplitNode is described in terms of PickSeeds and PickNext with no overhead. PickSeeds does need the private calculation of volumes. However, if all data is sent to one party (DNSP model), the multiplications to compute the volume will still be required.

In the case $K \geq 3$, the generation of a random number in a large enough range, the K data exchanges among the values of partial products masked by the random number, the initial masking (a multiplication) with the random number, and the division (by a random number) to obtain the result are all overhead. However, in terms of CPU operations, these are just 2 more floating point products. Because we have $K \geq 3$ dimensions, this is insignificant. What is costly is the k sequential data exchanges between the parties. On the other hand, PickNext incurs an overhead of 2 volume computations every time it is executed. It also costs as many as $\binom{M+1}{2}$ secure scalar product protocols (of vectors of dimension 2) in order to translate into a sum, plus one secure add vectors protocol for a vector of dimension $\binom{M+1}{2}$. For clustering, we may assume that all data items are inserted into an initially empty tree. The number of SplitNode operations is linear on the size of the tree. Thus, the number of SplitNode operations for a database with n items is $O(n/m)$. Note also that on insertion the algorithm ChooseLeaf has no privacy overhead except a volume calculation for each level when an entry whose box needs the least enlargement is sought. Again, classical results on analysis of algorithms show that in a tree that grows from empty to a balanced tree with $O(n/m)$ leaves, the cost of the executions of ChooseLeaf is $O(n \log(n/m)/m)$. Thus, the total overhead for all insertions is subquadratic in the number of nodes.

Now we analyse the overhead of the associative queries. Algorithm FindLeaf requires the public calculation of a conjunction of K Boolean values. Relative to the naive alternative of all the data, the cost is not the Boolean conjunction but the data exchanges of K Boolean variables in one site. Secure distance computations only occur at the leaves. Thus once again, the total overhead for all associative queries in linear on the number of nodes of the R-Tree. This analysis demonstrates that the time cost for the overhead caused by computations is perfectly reasonable and in fact much more affordable than running on one party with the union of the data from all parties (DNSP model). The concern may be the cost of data exchanges which is $O(kn/m)$ real values. However, if one takes into account that a solution that concentrates the data on one party needs to transmit $k-1$ databases of n records from the other parties to the data-concentration party, then one realizes that the privacy-preserving computation is in fact an even more efficient solution in terms of data exchanges.

Consider now the overhead in space. Each party has the structural information of the R-Tree. These are k copies of the R-Tree. However, each copy is only of the dimension that the data has at that party and therefore, it is only the pointer information that constitutes overhead (relative to the space that the concentration party would require to store in an R-Tree the data of all parties). The pointer size (i.e. space) is proportional to the external path length and thus also linear on the number of leaves in the R-Tree. Thus, the total overhead is $O(kn/m)$ pointers, which is a very reasonable price to pay for privacy. The last cost is for the generation of random numbers (this is also linear on the number of nodes). Hence, the total overhead is perfectly within the bounds of a practical efficient implementation.

4.2 Privacy-Preserving *KD*-Trees

Multidimensional binary search trees called K-Dimensional search trees or *KD*-Trees [53], are a generalisation of binary search trees to handle multidimensional points.

Definition 2 *A KD-Tree for a set of K-dimensional records is a binary tree, such that:*

1. *Each node contains a K-dimensional record and has an associated discriminant $j \in \{1, 2, ..., K\}$.*
2. *For every node with key $\vec{x}$ and discriminant j, the following invariant is true: any record in the left subtree with key $\vec{y}$ satisfies $y_j < x_j$ and any record in the right subtree with key $\vec{y}$ satisfies $y_j > x_j$.*
3. *The root node has depth 0 and discriminant 1. All nodes at depth d have discriminant $(d \bmod K) + 1$.*

A *KD*-Tree for a database $\mathbb{D}$ can be incrementally built by successive insertion into an initially empty *KD*-Tree as follows. The first key is put into a single node with two empty subtrees. Then, the first attribute of the second key is compared with the first attribute of the key at the root: if it is smaller, the second key is recursively inserted in the (empty) left subtree; otherwise, it is recursively inserted in the (empty) right subtree. Then, the first attribute of the third key is compared with the first attribute of the key at the root, and recursively inserted into the left or right subtree, as before. But if that subtree were not empty because it contained the second key, we would compare the respective second attribute of the second and third keys and proceed as before. In general, when inserting a key $\vec{x}$, we compare the key to be inserted with some key $\vec{y}$ at the root of some subtree: if $\vec{y}$ is at level j, we compare $x_{(j \texttt{ mod } K)+1}$ and $y_{(j \texttt{ mod } K)+1}$, and recursively continue the insertion in the left or the right subtree of $\vec{y}$, until a leaf (empty subtree) is found – recall that we assume that the key $\vec{x}$ is compatible with the tree where it is being inserted. Fig. 4.5 and Fig. 4.6 illustrate an example of *KD*-Tree structure.

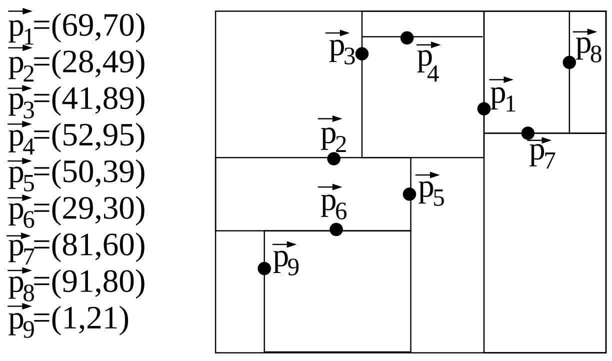


Figure 4.5: Partition of the search space.

Now let us see how we construct a K-dimensional search tree securely.

4.2.1 Vertically Partitioned Data

Without loss of generality, we will construct a secure solution for two parties and assume that the projection on each party is of one dimension. So, for

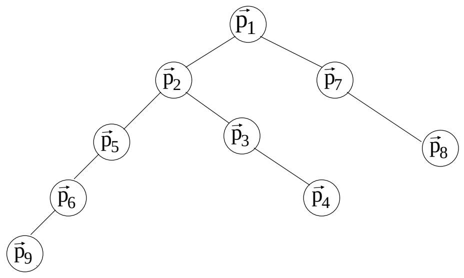


Figure 4.6: Corresponding $2D$-tree.

the moment, the number K of parties is 2 (as well as the dimension K of the attribute vectors). Extensions to more parties and cases where each party holds more than one attribute are easy to infer once we present the initial case. So, in what follows we have a setting where the data vectors look like $\vec{p} = (a, b)$. Because we assumed two parties and the data is vertically partitioned into one-dimensional domains for each party, we take it that the first party (Alice) knows the a values while the other one (Bob) knows the b values. All parties will know the structure of the tree; that is, all parties will know how many nodes are at each level and what records fall to the left or to the right on each internal node.

As the tree is constructed, both parties hold an empty tree. Both parties start by inserting the first point $\vec{p}_1 = (a_1, b_1)$. If later points (for example, consider the sequence of insertions $\vec{p}_2 = (a_2, b_2)$, $\vec{p}_3 = (a_3, b_3)$, and $\vec{p}_4 = (a_4, b_4)$) arrive, Alice compares the first coordinate a_1 with the first coordinates of the other points, namely a_2, a_3 and a_4. Alice registers and announces which records fall to the left of the root and which to the right. Those records with values greater than a_1 will go to the right leaf and those with values smaller that a_1 will go to the left leaf. Assume for illustration that $a_3 < a_1$, $a_2 > a_1$, and $a_4 > a_1$. Thus, points $\vec{p}_3$ will go to the left and point $\vec{p}_2$ and $\vec{p}_4$ will fall to the right. Now because we do not have any points in the left side except $\vec{p}_3$, insertions stop there, but will continue on the right side where two points are present.

The recursive insertion is then illustrated at depth 1 on the right side. First, $\vec{p}_2$ would be set as the key at this node. However, it is now Bob who will compare the second coordinate b_2 with the second coordinate of the point $\vec{p}_4$, namely b_4. Bob also announces the outcome of the comparisons so Alice knows also how the tree is progressing. For this example, the corresponding $2D$-tree is illustrated in

Fig. 4.7.

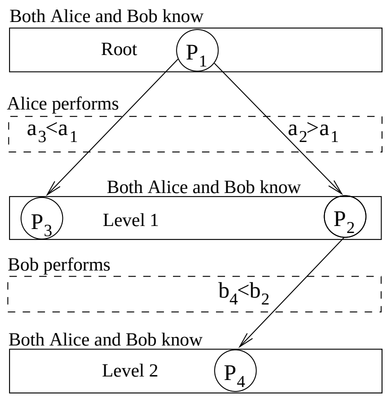


Figure 4.7: Example of $2D$-tree structure.

In order to obtain the $2D$-tree in every step we have to compare only two values (two coordinates) in the exclusive domain of one of the parties. Because data is partitioned vertically, each party knows all the values in its projection. So, each party can control the comparisons at a depth of the KD-Tree. In other words, parties only compare their own coordinates/information. Parties do not announce their values, only the outcome of comparison. Once this ownership of levels is set in the KD-Tree, all other algorithms for associative queries can be set up in the privacy-preserving KD-Tree.

The KD-Tree is a sufficient data structure for on small data sets that fit in RAM (main memory) and for which we are not performing many deletions and insertions of records (or locks for multi-user access or transactions) as it would be in the case of a multi-dimensional database. However, large databases that use concurrent access and use external disks (I/O to store on disk besides RAM) require R-Trees.

4.2.2 Horizontally Partitioned Data

Without loss of generality, we will construct a secure solution for two parties and assume that $K = 2$ (we are in 2D). So, for the moment, the number P of parties is 2 (as well as the dimension K of the attribute vectors). Extensions to more parties and cases where each party holds more than two attributes are easy to infer once we present the initial case. So, in what follows we have a setting where the data points look like $\vec{p} = (p_1, p_2)$ where both p_1 and p_2 are known by only one party. Because we have assumed two parties and the data is horizontally partitioned into two-dimensional domains for each party, we take it that the first party (Alice) knows a fraction of the total number of vectors while the other one (Bob) knows the other part (some number of vectors). All parties will know the structure of the tree; that is, all parties will know how many nodes are at each level and what records fall to the left or to the right on each internal node. The data at a node is only known by the party that owns the vector at that node. The other party only knows they are not the owner.

As the tree is constructed, both parties hold an empty tree. A random order is jointly decided on the totality of the data. Both parties insert the first vector; one will be the owner. Later points are included by recursive insertion. As a data vector arrives, there are two possibilities, the data point and the root of the subtree are owned by the same party. Then, that party performs the comparison and announces the result without any other party discovering here anything about the data values of the record being inserted or of the subtree's root. However, because data is partitioned horizontally, there would be cases where the vector being inserted and the subtree root node are owned by different parties. In this case, they can use Yao's comparison protocol [102] to compare the two values. Thus, parties never announce their values, only the outcome of comparisons. Once this secure comparison of levels is set in the *KD*-Tree, all other algorithms for associative queries can be set up in the privacy-preserving *KD*-Tree.

We hope that the *KD*-Tree illustration shows how this data structure could be shared with some level of privacy for associative queries. However, as we mentioned before, when the data structure is shared, the tree structure is known to all parties involved. The question is: How much can be inferred from this kind of information? Unfortunately, quite rapidly one party can learn bounding boxes on the data of the other party. It is not uncommon that on average, with a tree of depth 7, one party would have compared values with 4 of its data points. All data of other parties that fall in the bounding box determined by those four points are known to be inside that bounding box. According to the algorithm's output both Alice and Bob know the structure shown in Figure 4.8, but they

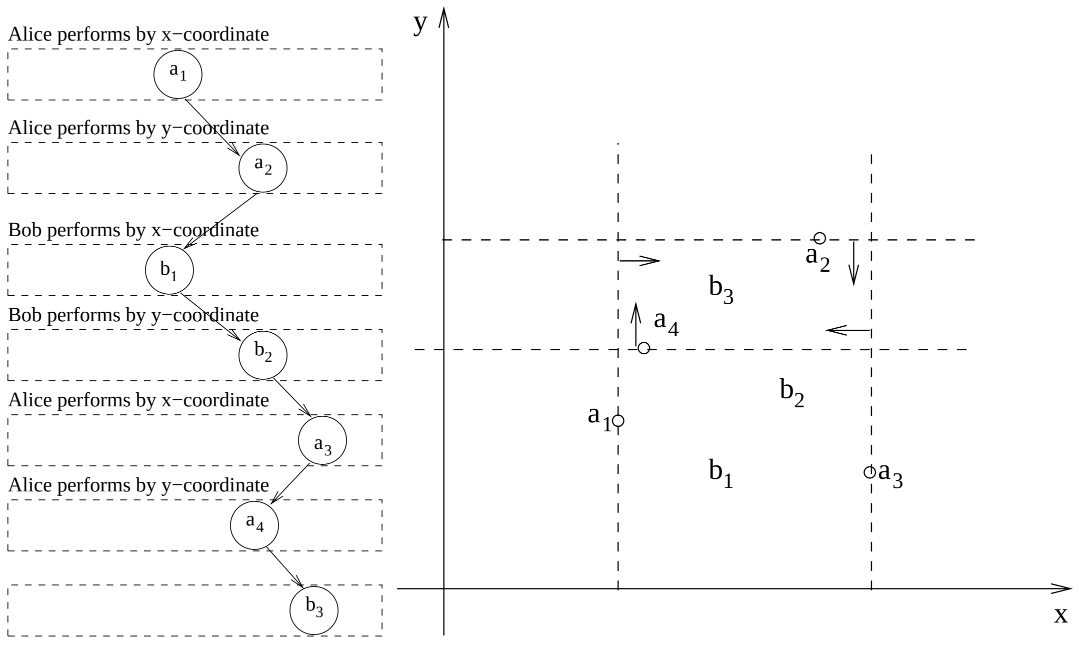

Alice performs by x−coordinate
a_1
Alice performs by y−coordinate
a_2
Bob performs by x−coordinate
b_1
Bob performs by y−coordinate
b_2
Alice performs by x−coordinate
a_3
Alice performs by y−coordinate
a_4
b_3
y
x
a_2
b_3
a_4
b_2
a_1
b_1
a_3

Figure 4.8: Example of partial $2D$-tree structure and how Alice obtains a range where $\vec{b_3}$ lies.

do not know the values of each other's points. Now let us see what Alice can infer from the output and her data about Bob's vector $\vec{b_3}$. Figure 4.8 shows how Alice obtains the approximate (range where the vector lies) location of $\vec{b_3}$. Here, because the *KD*-Tree is constructed by arranging the values to the left or to the right, according to the secure comparison, Alice using her values and an already constructed *KD*-Tree (she has as an output) detects the approximate location of the vector owned by Bob.

Of course with vectors of higher dimensions (more than 2) and more parties (more than 2), the level of privacy is higher. With K dimensions one party constructs a bounding box after having $K \times 2$ points in a path to some leaf.

4.3 Privacy-Preserving *SASH*

While we can consider the k-NN query solved for the privacy-preserving context by the PP k-NN algorithms, these algorithms should be used for small data sets, since their complexity is proportional to the number n of data vectors (see Section 3.3.1 and Section 3.3.4). For large data sets, we should use a data structure that allows much better performance on the number N of items in the database. To this end, we present a privacy-preserving *SASH*. This is because, as we have seen with R-Trees and KD-Trees, other data structures for associative queries usually induce a partition of the vector space and therefore represent boxes or boundaries where the data values of other parties are.

In this section we show a strategy by which we limit the number of distances that one party learns when the k-near neighbours to one of its data points is performed. The idea comes from approaches where the parties share a common data structure [39, 7]. Not that when parties share a data structure, they share the nodes and the pointers among the nodes, but not the information stored at the nodes. If we want to share a search data structure, but preserve privacy, we recommend the *SASH*. In this multi-level search structure queries are processed by recursively locating approximate neighbours within the sample, and then using the pre-established connections to discover neighbours within the remainder of the data set. We describe our privacy-preserving version of the algorithms of the *SASH* data structure.

For a privacy-preserving *SASH*, we must ensure that it is possible to implement all ADT-Dictionary operations (CONSTRUCT, INSERT, DELETE, SEARCH, etc.) and that each party will hold enough information to learn the desired output while being unable to discover data from records of other parties. The original *SASH* data structure considers a universe of n objects (not necessarily vectors) for which a similarity measure $dist(u, v)$ exists between any two objects u and v. A *SASH* is a directed edge-weighted graph with the following main properties.

- Each database object corresponds to a unique node. Little distinction will be made between a node, the object id or the database object to which it corresponds.

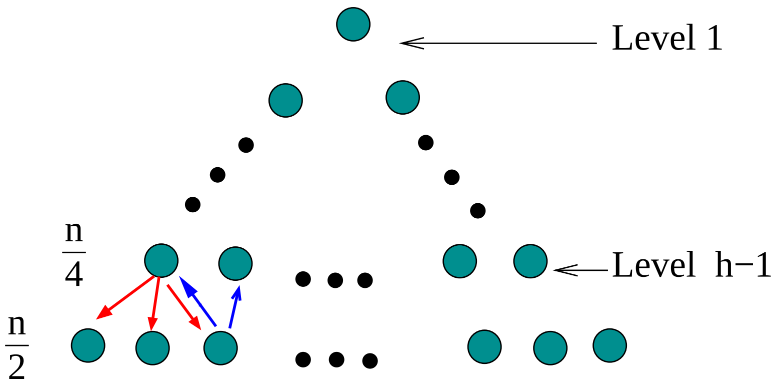
- The nodes are organised into a hierarchy of levels, ranging from a bottom level containing $\lfloor n/2 \rfloor$ nodes (the leaves), to a top level containing a single node (the root). With the possible exception of the top level, each level contains half as many nodes as the level below it, rounded down. The levels of the *SASH* are numbered from 1 to h.

- Edges within the *SASH* connect nodes from consecutive levels. Each node can have edges directed to at most p parent nodes as the level above it, and to at most c child nodes as the level below it. Every node except for the root must have at least one parent. The distance $dist(u, v)$ is always stored with each edge (u, v) at the time of its creation.
- Every node v (other than the root) has an edge directed to one parent $g(v)$ that is designated as its guarantor. The guarantor of v must have v as one of its children; v is called the dependent of $g(v)$. The requirement that every node have a guarantor ensures that every node is reachable from the root.

In our privacy-preserving *SASH*, all parties will know the entire graph structure of the *SASH*. That is, all parties will know how many nodes are at each level and what record correspond to each node. Each party knows who are the parents and children of a node as well as who is its guarantor or the dependent of a node. For illustration, consider for a moment two parties with database objects as vectors in $2D$ and each coordinate known only by one party. Thus, Alice holds the first coordinate and Bob holds the second coordinate of each record. While both may know that, say, the fifth record in the database corresponds to the root of the *SASH* neither will know the value of the other coordinate. The operations in the privacy-preserving *SASH* will inform all parties of the identifier of the record pointed by a node, its parents and its children, but will not reveal any values of the attribute-valued vector that describes the object.

4.3.1 Constructing the *SASH*

The edges of the *SASH* heuristically minimise the distances between their end-points. During the construction, each new node is attached to a small number of its near neighbours from the level above it. At the start of construction, the *SASH* is empty, and we insert all objects in a random and uniform order. We denote by $SASH_i$ the graph induced by the nodes from level 1 through i, for $1 \leq i \leq h$. Thus, $SASH_i$ is a *SASH* in itself. Iteratively constructing $SASH_1$, $SASH_2$, ..., $SASH_h$ results in the construction of the entire *SASH* (that is, $SASH_h$). The following algorithm shows how to construct $SASH_l$ given $SASH_{l-1}$ securely (for $1 \leq l \leq h$). This is where we add edges between nodes of the current last two levels.



The construction of entire *SASH* (that is $SASH_h$) proceeds
by iteratively constructing $SASH_1 \cdots SASH_h$ in order
$\Downarrow$
Development of the privacy-preserving k-NN algorithm is the
main task in adapting the *SASH* into privacy context

Figure 4.9: Illustration of the strategy used in the *SASH* data structure.

Algorithm Privacy_Preserving_Connect_SASH_Level(l):

1. If $l = 2$, then every node of level 2 will have the root node as its sole parent and guarantor, and the root node will have all nodes of level 2 as its children and dependents. This completes the construction of $SASH_2$. Note that, all parties must know the order of the data vectors (otherwise, the vertically partitioned data would not be able to join attributes for the same entity across parties). One can consider vertically partitioned data as several tables where a JOIN can be performed in a publicly known *id* attribute. Moreover, generating a random permutation of the object ids so that all parties know how to create the insertion order is no privacy risk. This would determine which entity is the root, the entities in level two and all the edges between these two levels.

2. Otherwise, for the remaining steps, we have $l > 2$. For each node v of level l, choose **securely** a set of up to p near neighbours $P_i(v, p)$ from among the nodes of each level $1 \leq i < l$. The algorithms for finding $P_i(v, p)$ uses $P_{i-1}(v, p)$, so we show how to compute $P_i(v, p)$ securely, assuming that $P_{i-1}(v, p)$ is computed securely (with a structural induction) as follows:

   (a) If $i = 1$, then $P_i(v, p)$ consists of a single node, the root (all parties know the id of the object here).

   (b) Otherwise, $i > 1$.

      i. Let $P'_i(v)$ be the set of distinct children of the nodes of $P_{i-1}(v, p)$. This can be obtained by all parties, because once $P_{i-1}(v, p)$ is computed securely, its children can be found using the shared knowledge of edges linking ids.

      ii. We compute $P_i(v, p)$ as the p nodes of $P'_i(v)$ closest to v, according to the measure *dist* (we use the PP-k-NN in Section 3.3.4). If $|P'_i(v)| < p$, then set $P_i(v, p) = P'_i(v)$.

3. We assign the parents of v to be the nodes of $P_{l-1}(v, p)$ in all replicas of the *SASH*. Recall that we consider vertically partitioned data as disjoint private tables where one common attribute (the ids) is public. The public edges of the *SASH* can be considered as linking id (and we continue to make no distinction between the nodes of the *SASH*, the objects, and the ids, except that the ids are keys for each party to the private attributes it holds). Each element v at level l now has up to p distinct parents associated with database elements in its vicinity.

4. Now, we create the child edges for the nodes of level $l - 1$, as follows:

   (a) For each node u of level $l-1$, each party determines the list of distinct nodes $C(u)$ of level l that have chosen u as a parent.

   (b) Using our privacy-preserving comparison of *dist* values, and the PP-k-NN with k set to c and the set of ids being $C(u)$, each party securely obtains the list to hold the c elements closest to u.

   (c) Now, each party connects these c nodes in $C(u)$ as the children of u.

5. Using the edges between ids, each party determines (for each node v of level l), whether it was accepted as a child of any node at level $l - 1$. If

a node was accepted, then the closest node that accepted it as a child becomes the guarantor $g(v)$ of v, and v becomes a dependent of $g(v)$. This guarantor is found by invoking our PP-k-NN with $k = 1$. If the node v was not accepted as a child, we label v as an orphan node.

6. For each orphan node v at level l, a node at level $l - 1$ is needed to act as its guarantor. The node should be as close as possible to v (in terms of the distance measure), and must be unencumbered; that is, it must have fewer than the maximum allowed number of children, c of children. Find a guarantor for v by successively doubling the size of the candidate parents set as follows:

   (a) Set $i = 1$

   (b) Compute $P_{l-1}(v, 2^i p)$ securely as in Step 2.

   (c) If $P_{l-1}(v, 2^i p)$ has no unencumbered node, let i++ and go to Step 6b.

   (d) Otherwise, choose as the guarantor $g(v)$ the unencumbered node of $P_{l-1}(v, 2^i p)$ that is closest to v (again, using PP-k-NN, with $k = 1$ on the set of unencumbered nodes). The parties add v as a child and dependant of $g(v)$, and replace the parent of v furthest from v by $g(v)$.

The previous discussions (regarding SMC metrics and PP-k-NN as building blocks, and the algorithms of the *SASH* construction) confirm that we can produce a privacy-preserving *SASH*. It it interesting how k-NN search is implemented. It may seem recursive, that in order to compute associative queries on a data structure, we found that we need k-NN as a SMC operation. And then, the data structure is to be used to find k-NN. The idea is that all internal k-NN are performed on small sets of vectors. This helps to constract the *SASH*, that we can use for efficiently performing k-NN on large databases.

This completes the construction of the privacy-preserving $SASH_l$. Note that the information shared by the parties is all the edges (parent/child relationships) and results of k-NN queries that identify only owners of vectors but do not reveal the data associated with those vectors. It may be necessary to demonstrate to all other parties that all the local data is involved in the process.

Note that, the *SASH* does not partition the search space, as a *KD*-Tree or an *R*-Tree does. As we have seen in our case study *KD*-Trees (see Figure 4.8 in Section 4.2.2), partitioning the space may reveal a bounding box for some vectors. Although not showed here, but this applies to *R*-Trees as well, since in fact, a node in an *R*-Tree is a bounding box for all data below that node.

4.3.2 Private Approximate k-NN Queries.

A simple and effective way to retrieve an approximation to the k-near neighbours of a query object q is to generate the candidate parents as in the *SASH* construction. This allows us to compute $P_1(q,k) \cup P_2(q,k) \cdots \cup P_h(q,k)$ and then we select k elements closest to q as the result of the query. Since all nodes are reachable from the root, there is a level j with more than k elements where $|P_j(q,k)| = k$. Thus, exactly k elements will be returned, (provided that the number of elements in the database is at least k).

The authors of the *SASH* propose a search pattern that improves both accuracy and search time [60] (a variable number $k_i = \max\{k^{1-\frac{h-i}{\log_2 n}}, \frac{1}{2}pc\}$ of objects is drawn from each level $1 \leq i \leq h$). The number of objects k_i selected from level i does not depend on the query object q. We can implement this privacy-preserving variant as follows.

Algorithm Privacy_Preserving_FindNearNeighbors(q,k):

1. Construct securely a set of up to $k_i > 0$ near neighbours $P_i(q,k_i)$ from among the nodes of *SASH* level i, for $1 \leq i \leq h$, as follows:
   (a) If $i = 1$, then $P_i(q,k_i)$ consists of a single node, the root (this is public knowledge).
   (b) Otherwise, $i > 1$.
      i. Let $P_i'(q)$ be the set of distinct children of the nodes of $P_{i-1}(q,k_i)$ (these are publicly known edges of the *SASH*).
      ii. Set $P_i(q,k_i)$ to be the k_i nodes of $P_i'(q)$ closest to q using our PP-k-NN with $k = k_i$ on the set $P_i'(q)$. If $|P_i'(q)| < k_i$, then set $P_i(q,k_i) = P_i'(q)$.

2. Using PP-k-NN again, return the k elements of $P_1(q, k_1) \cup P_2(q, k_2) \cdots \cup P_h(q, k_h)$ closest to q. If the set contains fewer than k elements, return the entire set.

4.3.3 Private Range Queries.

The *SASH* can also be used to perform approximate range queries by iteratively computing approximate k-NN queries for some increasing sequence of value $k = s_1,\ s_2,\ s_3,\ \cdots$. For example, the size of the query could be doubled at each iteration ($s_{i+1} = 2s_i$ *for* $i > 1$). The iteration would continue until either an element outside the desired range is discovered (at which time all generated elements that lie within the range are reported as the solution to the range query), or the entire database has been visited (which occurs only when most or all of the database elements lie within the query range). If we use this doubling strategy, we can guarantee a competitive ration of two; namely, the final value k is guaranteed to be at most twice the true number of elements lying in the desired range. Because range queries are based upon privacy-preserving approximate k-NN queries, (i.e. we use approximate k-NN queries for some increasing sequence of value $k = s_1,\ s_2,\ s_3,\ \cdots$), this immediately means that constructing a *SASH* and performing approximate k-NN queries in the privacy-preserving context suffices to have range queries in the privacy-preserving context.

While we have not shown Delete or other ADT-Dictionary operations here, the description on insertion/construction should suffice to perform the necessary extensions.

4.3.4 Performance and Experimental Results

For the overwhelming majority of methods for k-NN queries and associative queries, the major cost is not the computation of distances per se, but how many of these computations are performed. That is, as long as computing distances are proportional to the number of dimensions, the cost (associated with k-NN queries or associative queries) is essentially the number of evaluations of distances. The only possible competitor to our privacy-preserving *SASH* method to perform

	Average Size S of the union for $A0$ algorithm			
	number m of parties			
k	4	6	8	10
2	2,469±259	2,454±197	3,503±213	4,377±178
3	2,589±243	2,559±194	3,729±173	4,595±142
4	2,548±249	2,894±192	3,903±172	4,645±166
5	2,753±254	3,156±168	4,007±157	4,806±145
8	2,891±228	3,516±151	4,334±143	5,052±137
10	2,982±218	3,694±138	4,535±136	5,092±117
15	3,173±191	3,973±121	4,708±121	5,314±97
20	3,100±203	4,037±141	4,835±106	5,438±55
25	3,429±161	4,270±127	5,004±98	5,448±69
50	3,704±150	4,689±125	5,294±68	5,628±44

Table 4.1: Evaluation of the A0 algorithm on Database 1 (The Insurance Company Benchmark CoIL 2000).

k-NN queries is the privacy-preserving version [95] of Fagin's A0 algorithm [48]. However, for a vertically partitioned database with N records, this algorithm's complexity (time and communication cost) includes as a factor the number S of candidates generated. It is well recognised that S can be as large as N and in the best case as small as k. The accepted [48, 95] worst-case theoretical analysis is that the complexity is $O(N^{(m-1)/m}k^{1/m})$ where m is the number of parties.

However, there are no studies on what is the expected performance of this algorithm. Our intuition is that Fagin's algorithm must perform poorly in general because in order to perform well it requires that the cylinders around the query vector $\vec{q}$, that constitute the projection to the k-nearest neighbour in each party, contain together as few elements as k. This seems unlikely. To confirm this we evaluated the size S of the union of Fagin's AO algorithm in 5 well-known large data sets. The CoIL 2000 Challenge [97] dataset (Database 1) contains information on customers of an insurance company. The data consists of 86 variables and includes product usage data and socio-demographic data derived from zip area codes. We repeated the following experiment 100 times. We partitioned the attributes randomly into m parties, we selected random metrics for each party (among Euclidean, Hamming, Chessboard and *Minkowski* with $r = 1$), we selected a random query point from the data and computed the k-nearest neighbours using Fagin's A0 algorithm. Table 4.1 shows the average size S of the union in Fagin's AO algorithm for this data set with 95% confidence intervals. This data set has 5, 822 records and we can see that most of the entries in the table are close to or above 3, 000 while several are above 5, 000. It is rather disappointing that when asking for 10 neighbours among 8 parties we expect a union size to be 78% of the size N of the database. We also recorded the best and worst observed size S of the union. Rather than showing another table we present this data for $m = 8$ parties in Fig 4.10. Note that the worst case for all

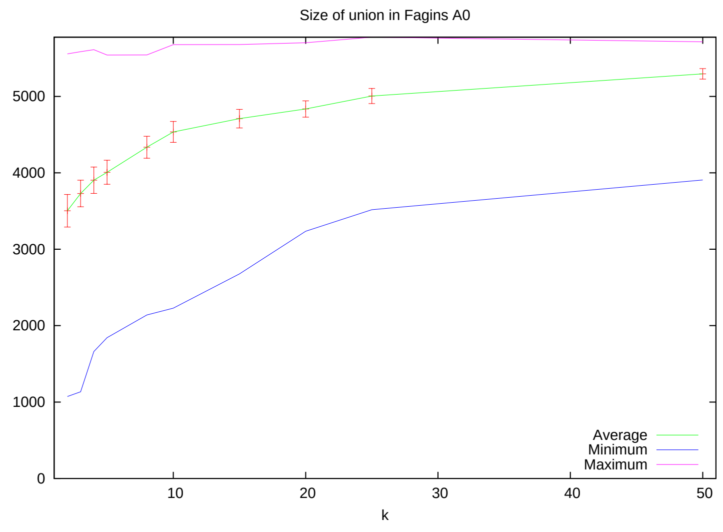


Figure 4.10: Maximum, Average and Minimum size of S with $m = 8$ parties for Database 1.

query sizes k is above 5, 000 and that the size of the union in the best observed case is well above $500 \times k$, and rapidly above 50% of the size of the file. Similar results occur for the Census-Income Database holding multivariate PUMS census data (Database 2) from the Los Angeles and Long Beach areas for the years 1970, 1980, and 1990 (from KDD UCI repository). Combining test and training databases we get 299,285 records with 40 dimensions/attributes.

The inefficiency of A0 is also reflected in three large datasets previously used for approximate nearest neighbour queries [51].

The dataset named *Histogram* (Database 3) corresponds to a colour histogram, while the one named *Stock* (Database 4) corresponds to a stock market price. *Stock* has dimension 360 and 6,500 records corresponding to different companies.

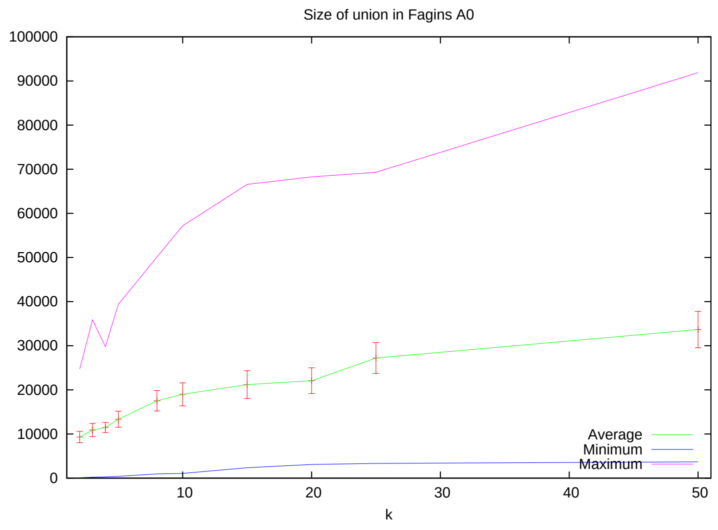


Figure 4.11: Maximum, Average and Minimum size of S with $m = 4$ parties for Database 2.

For $m = 8$ parties, the observed average, minimum and maximum size of the union for AO are shown in Fig 4.12 for the *Stock* dataset while the results for *Histogram* dataset are shown in Fig. 4.13. The histogram data set has dimension

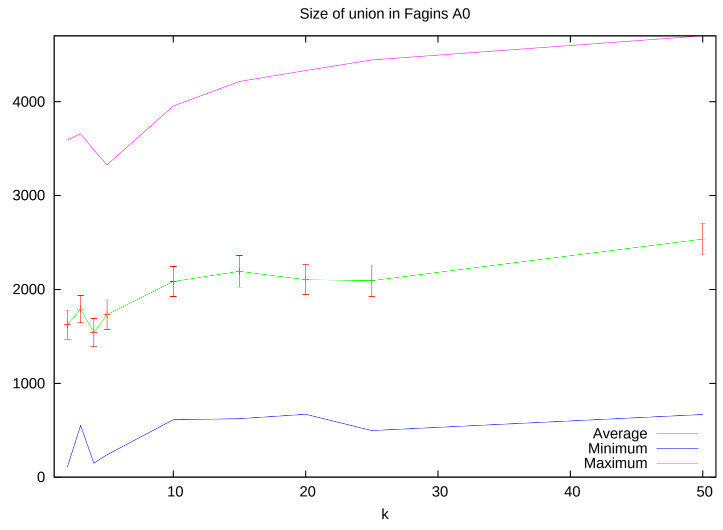


Figure 4.12: Maximum, Average and Minimum size of S with $m = 8$ parties for Database 4.

64 and 12,103 records. The results for *histogram* show not only an average case of $O(n)$ but the worst case is essentially N. Finally, an aerial image dataset with dimension 60 and 275,465 (Database 5) records was tested and results for $m = 8$ on the size of the union for AO appear in Fig. 4.14. Another reason

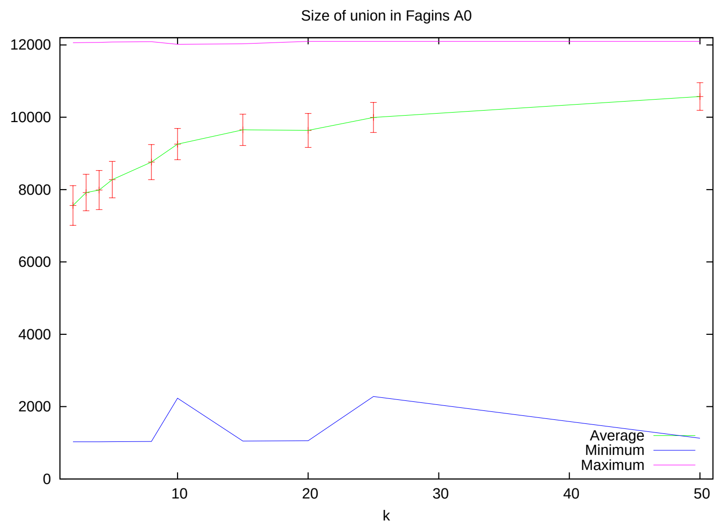


Figure 4.13: Maximum, Average and Minimum size of S with $m = 8$ parties for Database 3.

for performing this analysis is that the privacy-preserving version of Fagin's AO algorithm [95] leaks all the ids of the union. Therefore, the size S of the union not only determines the inefficiency of the method but is also a strong measure of the lack of security in the algorithm.

The privacy-preserving AO algorithm will perform at least as many distance evaluations as the number S of candidates (or the size of the union).

We have chosen the *SASH* because this data structure is very efficient invoking a number of distance computations which is bounded by $pcN \log 2N$ [61, 60] (for construction), while the bound for an approximate k-NN query under the uniform search is $ck \log 2N$, and $\frac{k^{1+\frac{1}{\log 2N}}}{k^{\frac{1}{\log 2N}} - 1} + 2p^3 \log 2N$ for geometric search (here p and c are the constant parameters of the *SASH* and k is the number of NN requested). Therefore, the *SASH* will easily outperform Fagin's A0 algorithm, and will provide logarithmic response for k-NN queries and associative queries.

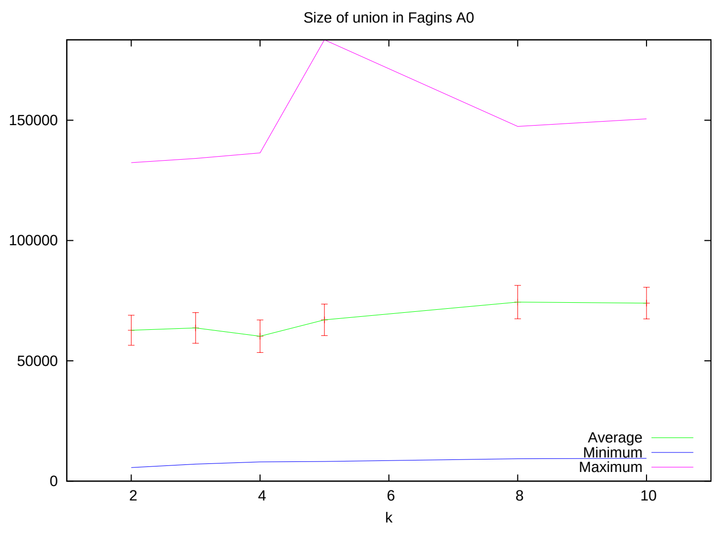


Figure 4.14: Maximum, Average and Minimum size of S with $m = 8$ parties for Database 5.

Naturally, the *SASH* invokes k-NN queries on small data sets. That is, the privacy-preserving *SASH* invokes Protocol 10 for small sets of size n. The natural question is why not use the searching for top k-queries in a field which requires $(n+1)\log|F|$ rounds, as opposed to our Protocol 10 which has complexity $O(n)$.

First, clearly all steps are equivalent until these protocols receive two vectors of dimensions n known to two distinct parties. Our protocol does require $\Theta(n \log n)$ time on Alice's side for sorting and $O(n)$ time on Bob's side to add a random value to all the shares it holds and to generate the random permutation π, but the constants involved are small and clearly the process is practical. However, the search in a field requires $(n+1)\log|F|$ rounds. Typical values are $F = 10^6$, which makes a larger constant in the $O(n)$ for this alternative.

More importantly, each of these rounds involves a Yao-comparison with shares and a final round where each party totals n values. Clearly, the local computation is far more in this aspect alone than our algorithm. In terms of communication cost, our approach is also more efficient. Bob sends exactly n values, and Alice sends back k values. The binary search in a field performs the communications needed for $(n+1)\log|F|$ Yao-comparisons.

To further illustrate the practicality of our approach, we have implemented our Protocol 10. Overall, Protocol 10's complexity depends on the number m of parties, the number n of vectors and the number k of near neighbours we are looking for.

In Figure 4.15, we illustrate this dependency. The implementation confirms the logarithmic performance time for m and n. The dependency from k oscillates (in a small region) but it remains bounded by constant. This is clear, because the sorting component is $O(n \log n)$ while the selection of k values is $O(k)$, with k much less than n. For communication cost, the $mn + k$ complexity is dominated by n again.

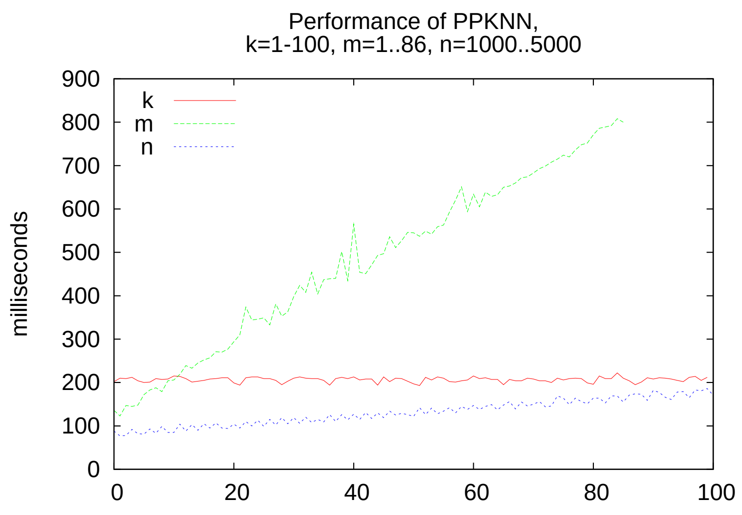


Figure 4.15: Dependency on k, m and n tested on Database 1 - The CoIL 2000 Challenge dataset

We have also implemented the *SASH* method and evaluated the performance on the same 5 databases used for Fagin's A0 algorithm. The performance depends directly on the number of candidates generated at all levels of the *SASH* (the sum of $\|P_i(q, k_i)\|$ for all levels [60, page 8]). Our results for all 5 databases are shown in Table 4.2. The *SASH* candidate generation does not depend on m, and shows remarkably small numbers for all data sets.

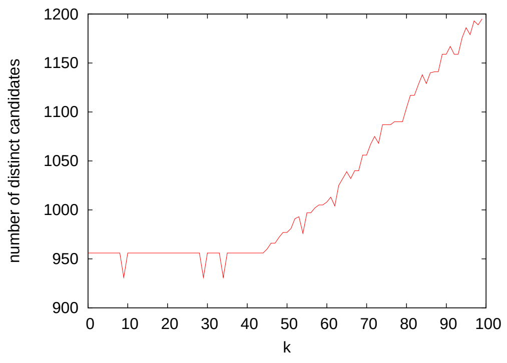
	Size of the union for the *SASH* method for all 5 databases				
k	database 1	database 2	database 3	database 4	database 5
2	956	1506	1229	739	1438
3	956	1506	1229	739	1438
4	956	1505	1229	739	1438
5	956	1505	1229	739	1438
8	956	1506	1229	739	1438
10	931	1505	1229	739	1438
15	956	1506	1229	739	1438
20	956	1505	1229	739	1438
25	956	1506	1229	739	1438
50	977	1512	1255	747	1463
75	1087	1647	1391	815	1582
100	1195	1796	1552	889	1738

Table 4.2: Number of distinct candidates generated during k-NN queries using *SASH*.

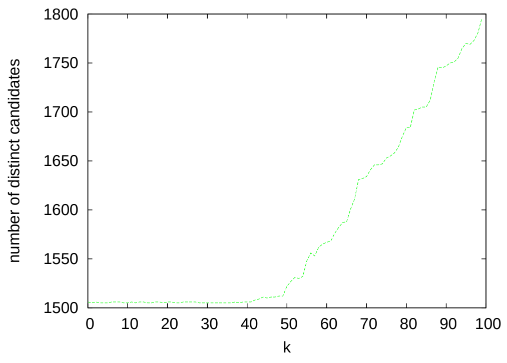
We have used the default parameters for *SASH* recommended by the authors [3]. That is, we set the maximum number p of parents per node is 4 and the maximum number c of children per node is $4p$ (that is 16) [60]. The geometric search pattern [60] is the one used for k-NN queries. The impact of this geometric pattern in our experiments is noticed in the Figures 4.16(a), 4.16(b), 4.17(a), 4.17(b) and 4.18(a). In particular, across these figures, the number of generated condidates remains essentially constant for k below $k = 50$. This is clear, because during the construction of the *SASH* the parameters p and c determine the size of the local collection of information about closest near neighbours for each node (see Sections 4.3.1 and 4.3.2). In particular, this influences the values of the geometric pattern used for k-NN query. In this geometric pattern, a variable number $k_i = \max\{k^{1-\frac{h-i}{\log_2 n}}, \frac{1}{2}pc\}$ of objects is drawn from each level $1 \leq i \leq h$). Let us examine, for instance, Figure 4.17(a). In this dataset, the vale $n = 12,103$ and during the construction of the *SASH*, a total of 11 levels are contracted, so $h = 11$. When $k \leq \frac{1}{2}pc = 32$, then every $k_i = 32$ for $1 \leq i \leq h$. If $32 \leq k \leq 42$, then in the last level ($i = h$) only the number of generated candidates will be different, but this will not affect the outcome as much, due to the very small

[3]We thank Dr. Michael Houle for providing two initial implementations of the SASH from which we were able to create our experiments.

difference. Moreover, the generated extra candidates could not be distinct from the candidates already included in the list. This is exactly what happens here. Starting from $k > 42$ the last two levels produce more distinct candidates, so an increase in the overall number of generated candidates is noticeable (see Figure 4.17(a)). An increase of the value of k will affect more levels of the *SASH*. In particular, for every node from a level above the affected levels, the number of distinct children sought will be more than $\frac{1}{2}pc = 32$. This, obviously, affects the overall result.

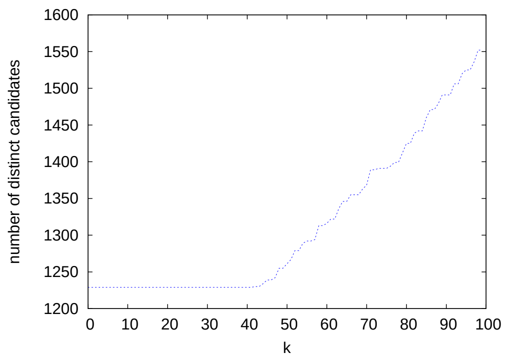


(a) Database 1

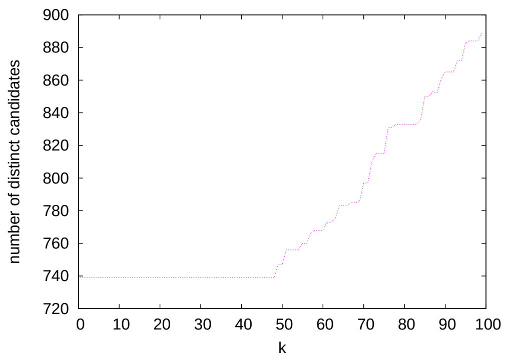


(b) Database 2

Figure 4.16: Distinct candidates generation while performing k-NN queries for Database 1 and Database 2.



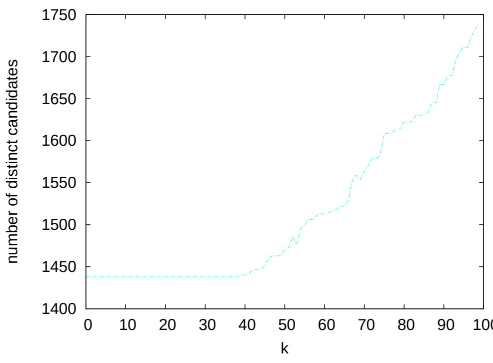
(a) Database 3



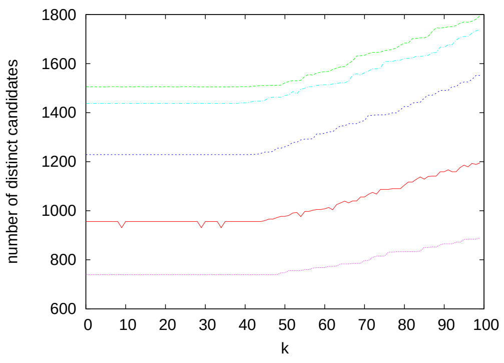
(b) Database 4

Figure 4.17: Distinct candidates generation while performing k-NN queries for Database 3 and Database 4.

We alert the reader that the scale of the $y - axis$ in Figures 4.16(a), 4.16(b), 4.17(a), 4.17(b) and 4.18(a) is logarithmic. The Figure 4.18(b) collates all the



(a) Database 5



(b) All 5 databases

Figure 4.18: Distinct candidates generation while performing k-NN queries for Database 5 and all 5 databases.

	Fagin's PP A0	**SASH**
Average number of candidates leaked for k-NN queries	Very close to N (the size of the database)	$\Theta(\log N)$ (logarithmic on the size of the database)
Performace	Very close to N (the size of the database)	Sublinear (depending on the parameters c and p of the *SASH*, but not guaranteed to be logarithmic)

Table 4.3: Comparison of the Fagin's A0 and *SASH*algorithms.

data for the *SASH* with logarithmic performace.

Thus, the performance of *SASH* candidates is much better than the performance of Fagin's A0 candidates (at least several orders of magnitude, see Table 4.3). This demonstrates the efficiency of the *SASH* approach.

Chapter 5

Regression and Clustering

In this chapter we focus on Regression and Clustering since these are the overlaps between the field of statistics and the field of data mining, and what seems to constitute the most applied and discussed approaches in data analysis.

5.1 Regression Analysis

Regression is arguably the most applied data analysis method. Today there are many scenarios where data for attributes that correspond to predictor variables and the response variable itself are distributed among several parties that do not trust each other. Privacy-preserving data mining has grown rapidly studying the scenarios where data is vertically partitioned. While algorithms have been developed for many tasks (like clustering, association-rule mining and classification), for regression, the case of only two parties remained open. Also open is the most interesting case when the response variable is to be kept private.

Regression analysis examines the relationship of a dependent variable Y (the *response variable*) to specified independent variables (the *predictors*). Regression is a fundamental tool in data analysis used for prediction, modeling of casual relationships, and scientific hypothesis testing about relationships between variables, among many other uses. The mathematical model of the relationship is the regression equation. *Linear regression* is a regression method that explores a linear relationship between the dependant variable Y and the k independent variables x_i (allowing an error term ϵ). That is, the form of the model (the regression equation) is $Y = \alpha + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_k x_k + \epsilon$. Linear regression is the most studied regression approach. The coefficients $\alpha = \beta_0$ and β_i are the parameters learned from the data.

In order to have privacy-preserving linear regression one must perform several steps which include secure calculation of the regression coefficients β_i and model diagnostics. The calculation of the regression coefficients is an important step in regression but the diagnostics and model selection are even more important and challenging. Diagnostics checks whether a model is proper and the best possible, or whether it needs revision by further analysis. This can be carried out by graphical tools that include plotting the *residual versus the predicted response* and/or *residual versus predictor* plots. Model selection can be performed iteratively, controlled by the analyst based on diagnostics analysis, or automatically by stepwise regression, or exhaustively, that is, running over all possible models relying on some model selection criteria such as Mallow's C_p statistic.

The first solution for linear regression in the privacy context [38] was based on a series of protocols for matrix multiplication that were secure in a weak sense [96]. This was an extention of secure two-party statistical analysis presented before [36]. Other solutions addressed the simpler case of horizontally partitioned data [69]. Using Powell's algorithm for solving quadratic minimization an alternative was provided for the vertical partitioning case [86]. All of these assume that the response variable Y is known to all the parties. This reduces the cases where two different parties may attempt to understand the relationship between attributes in their data by means of linear regression. Specially, if there is no commonly known attribute that can act as the commonly known response. For example, one company may hold salary and education level for a large set of employees, while the medical insurer may hold data about the frequency of medical checkups. It would be difficult to explore the relationship between education level and the monitoring which individuals perform on their health. Similarly, the potential relationships between types of treatment and professional activity (that could lead to patterns in certain conditions due to the nature of the job). Similarly, if there was a third party, say a retailer, then it may be interesting to explore the types of expenses in relation to salary. This may enable the retailer to target its advertising, and the employer to offer some employment benefits according to some rank within the organization.

Although the previous solutions [38, 86, 12] provide privacy-preserving calculation of regression coefficients for two parties holding **more than one attribute**, Vaidya and Clifton have remarked [96] on a potential privacy breach of the attribute values when using the residual versus predictor plots to determine whether the fitted model is proper. For example, Alice can generate the residual versus x_1 plot. In the plot, the coordinates of the points are exactly the values of x_1. If the plot is revealed to the other party (Bob), Bob may use the plot to recover accurate values of x_1 which are held by Alice. We will show that this situation can be avoided if Alice and Bob distribute residuals in private shares and apply secure multi-party computation protocols for diagnosis analysis. But none of the solutions in the literature provides coefficients as well as residuals with shares.

Another privacy risk that was highlighted previously [96], is the case of only two parties, each holding only one attribute. In this scenario, the disclosure of the residuals immediately results in the disclosure of the attribute values of the opposite party. We will also provide the first solution to this case.

5.1.1 Privacy-Preserving Bivariate Linear Regression

Bivariate linear regression models the response variable Y as a linear function of just one predictor variable X; that is $Y = \alpha + \beta X + \epsilon$, where α and β are regression coefficients specifying the Y-intercept and slope of the line, respectively. These coefficients can be found by minimisation of the error ϵ between the actual data and the estimate of the line. Given n sample data points of the form (a_1, b_1), $\cdots$, (a_n, b_n), then the regression coefficients estimated by the method of least squares are

$$\beta = \frac{\sum_{i=1}^{n}(a_i - \bar{a})(b_i - \bar{b})}{\sum_{i=1}^{n}(a_i - \bar{a})^2} \quad (1) \qquad \text{and} \qquad \alpha = \bar{b} - \beta\bar{a} \quad (2)$$

where $\bar{a}$ is the average of $a_1, \cdots, a_n$ and $\bar{b}$ is the average of $b_1, \cdots, b_n$.

When data is vertically partitioned, Alice will know all $a_1, \cdots, a_n$ and Bob will have $b_1, \cdots, b_n$. Thus, Alice and Bob can calculate each $\bar{a}$ and $\bar{b}$ without any communication. The goal would be for Alice and Bob to obtain the coefficients for $Y = \alpha + \beta X$, while they do not learn each other's data points. Note, however, that knowledge of β and α by Alice and Bob implies (because $\alpha = \bar{b} - \beta\bar{a}$), that each will learn something about each other's data. Alice will discover $\bar{b}$ and Bob $\bar{a}$. It is commonly accepted in secure multi-party computation that anything that can be learned from the output $f(\vec{x}, \vec{y})$, about the other party's data is acceptable. We will present this situation first. The alternative, is that the output $f(\vec{x}, \vec{y})$ is in shares. We will present this case second.

In order for the parties to find β, we provide the following protocol. First note that the dividend in Eq. (5.1) is a scalar product of two vectors, each known to one party only.

$$\begin{aligned}&\sum_{i=1}^{n}(a_i - \bar{a})(b_i - \bar{b}) \\ &= \big((a_1 - \bar{a}), \cdots, (a_n - \bar{a})\big)^T \cdot \big((b_1 - \bar{b}), \cdots, (b_n - \bar{b})\big). \qquad (5.3)\end{aligned}$$

Using the SCALAR PRODUCT PROTOCOL WITH PRIVATE SHARES [1] but providing an answer to Alice only, she can then divide by $\sum_{i=1}^{n}(a_i - \bar{a})^2$ (which she owns) and obtain β. Alice would then pass β to Bob. In this way Alice and Bob learn the coefficients with a protocol that is the best possible in the sense that the protocol reveals to each party the final result and only what can be discovered using the final result and one's input. This solution is similar to the one presented before [36].

However, if the coefficients are to be learned in shares $\alpha = s_a(\alpha) + s_b(\alpha)$ and $\beta = s_a(\beta) + s_b(\beta)$ (with $s_a(\alpha), s_a(\beta)$ known only to Alice and $s_b(\alpha), s_b(\beta)$ known

[1] In this case Bob sets his private share $V_2 = 0$ (see Section 2.7)

only to Bob) we need additional care. By Eq. (5.2), if $\beta = s_a(\beta) + s_b(\beta)$, then

$$\begin{aligned}\alpha &= \bar{b} + [s_a(\beta) + s_b(\beta)]\bar{a} = \bar{b} + s_b(\beta)\bar{a} + s_a(\beta)\bar{a} \\ &= \left(1, \bar{a}, \bar{a}s_a(\beta)\right)^T \cdot \begin{pmatrix} \bar{b} \\ s_b(\beta) \\ 1 \end{pmatrix}.\end{aligned}$$

Because the first vector above is known only to Alice and the second is known only to Bob, using the SCALAR PRODUCT PROTOCOL WITH PRIVATE SHARES would provide the required $s_a(\alpha)$ for Alice and $s_b(\alpha)$ for Bob with

$$\alpha = s_a(\alpha) + s_b(\alpha).$$

Thus, providing the coefficients with shares reduces to providing β with shares. We accomplish this requirement as follows. Recall that β is an expression (Eq. (5.1)) whose dividend is a scalar product (Eq. (5.3)). Thus, we can obtain the dividend as two values A_1 and B_1, with A_1 only known to Alice and B_1 only known to Bob. Let Alice generate a random number R, that she passes to Bob, and consider the following derivation.

$$\beta = \frac{A_1 + B_1}{(\sum_{i=1}^{n}(a_i - \bar{a})^2 + R) - R} = \frac{A_1 + B_1}{A_2 + B_2}.$$

This has the form of the division protocol with A_1, A_2 only known to Alice, and although B_1 is only known to Bob, the value $B_2 = R$ is known to both Bob and Alice. Nevertheless, we can apply the SECURE DIVISION PROTOCOL from Section 2.11. Knowledge of $B_2 = R$ by Alice results in Alice learning r_2, but interestingly enough, this is still insufficient for Alice to learn b_1 or Bob's share in the output [2]. This gives then the required shares for β.

We have provided two protocols. Firstly, Alice and Bob learn the coefficients α and β. In the second one, they learn these coefficients, but in shares. Both protocols are ideal, in the sense of privacy from the semi-honest model in SMC, as what each party learns about the other's data is nothing more than what can be inferred from the specified output of the protocol and its own data.

[2]Full proof of this requires description of the SCALAR PRODUCT PROTOCOL WITH PRIVATE SHARES which one can see in the Section 2.7.

5.1.2 Privacy-Preserving Multiple Regression

In this section we investigate multiple regression. Here, several parties are involved with several attributes and the goal is again to obtain linear regression coefficient across different attributes. We once more do not assume a response variable in common. In fact, this enables cases where the response variable is an attribute known by one of the involved parties. This provides the ability to find relationships between different attributes of different parties. The only assumption now is that there are three or more non-virtual parties. ($m \geq 3$).

Using the least squares method, the vector of coefficients $\vec{\beta} = (\beta_1, \beta_2, \cdots, \beta_m)$ is $\vec{\beta} = (X^T X)^{-1} X^T \vec{Y}$, where the matrix $X = (\vec{X}_1, \vec{X}_2, \cdots, \vec{X}_m)$ has column vectors $\vec{X}_j$ and each $\vec{X}_j$ is owned by $j - th$ party. The response variable is a vector $\vec{Y}$ owned by one party only.

The task here is to compute β without revealing any party's data. We first present a protocol that reveals $\vec{\beta}$ to all parties.

Let us first describe how to compute $X^T X$. We start with the three-party case, ($m = 3$). If we have n data points of the form (a_i, b_i, c_i) with a_i known to Alice only, b_i known to Bob only and c_i known to Charles only, we have the following data matrix X given by

$$X^T = \begin{pmatrix} a_1 & a_2 & a_3 & \dots & a_n \\ b_1 & b_2 & b_3 & \dots & b_n \\ c_1 & c_2 & c_3 & \dots & c_n \end{pmatrix}.$$

Then, by using the SCALAR PRODUCT PROTOCOL WITH PRIVATE SHARES, whenever we have a matrix entry with data vectors belonging to two different parties, we obtain the following derivation[3] for the symmetric matrix X^TX.

$$\begin{aligned}
&X^TX \\
&= \begin{pmatrix} \vec{a}^T \cdot \vec{a} & \vec{a}^T \cdot \vec{b} & \vec{a}^T \cdot \vec{c} \\ \vec{b}^T \cdot \vec{a} & \vec{b}^T \cdot \vec{b} & \vec{b}^T \cdot \vec{c} \\ \vec{c}^T \cdot \vec{a} & \vec{c}^T \cdot \vec{b} & \vec{c}^T \cdot \vec{c} \end{pmatrix} \\
&= \begin{pmatrix} \vec{a}^T \cdot \vec{a} & V_{ab}^A + V_{ab}^B & V_{ac}^A + V_{ac}^C \\ V_{ab}^A + V_{ab}^B & \vec{b}^T \cdot \vec{b} & V_{bc}^B + V_{bc}^C \\ V_{ac}^A + V_{ac}^C & V_{bc}^B + V_{bc}^C & \vec{c}^T \cdot \vec{c} \end{pmatrix} \\
&= \begin{pmatrix} \vec{a}^T \cdot \vec{a} & V_{ab}^A & V_{ac}^A \\ V_{ab}^A & 0 & 0 \\ V_{ac}^A & 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & V_{ab}^B & 0 \\ V_{ab}^B & \vec{b}^T \cdot \vec{b} & V_{bc}^B \\ 0 & V_{bc}^B & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 & V_{ac}^C \\ 0 & 0 & V_{bc}^C \\ V_{ac}^C & V_{bc}^C & \vec{c}^T \cdot \vec{c} \end{pmatrix}.
\end{aligned}$$

Thus, X^TX can be essentially computed by the ADD VECTORS PROTOCOL, but instead of vectors we use matrices (also called SECURE SUM [96]). Here, each party owns one matrix.

In order to add them securely, Alice (the first party) can generate a random matrix and pass it to Charles (the third party) who will add his matrix to the random matrix received from Alice. Charles will send this sum to Bob. Next, Bob will add his matrix to the matrix received from Charles and send it to Alice. Alice subtracts the original random matrix and adds her matrix to obtain X^TX.

For the case $m \geq 3$ (with n data points), we use the SCALAR PRODUCT PROTOCOL WITH PRIVATE SHARES (see Section 2.7) to obtain $\vec{p}_i^T \cdot \vec{p}_j$ as $V_{p^ip^j}^i + V_{p^ip^j}^j$ when $\vec{p}_i, V_{p^ip^j}^i$ is known only to the i-th party and $\vec{p}_j, V_{p^ip^j}^j$ is known only to the

[3]The super-index provides the owner party.

j-th party, whenever $i \neq j$. Thus,

$$\begin{aligned}
X^T X &= \begin{pmatrix} p_1^1 & \cdots & p_n^1 \\ p_1^2 & \cdots & p_n^2 \\ \vdots & \ddots & \vdots \\ p_1^m & \cdots & p_n^m \end{pmatrix} \cdot \begin{pmatrix} p_1^1 & \cdots & p_1^m \\ p_2^1 & \cdots & p_2^m \\ \vdots & \ddots & \vdots \\ p_n^1 & \cdots & p_n^m \end{pmatrix} \\
&= \begin{pmatrix} \vec{p}_1^T \cdot \vec{p}_1 & \vec{p}_1^T \cdot \vec{p}_2 & \cdots & \vec{p}_1^T \cdot \vec{p}_m \\ \vec{p}_2^T \vec{p}_1 & \vec{p}_2^T \cdot \vec{p}_2 & \cdots & \vec{p}_2^T \cdot \vec{p}_m \\ \vdots & \vdots & \ddots & \vdots \\ \vec{p}_m^T \cdot \vec{p}_1 & \vec{p}_m^T \cdot \vec{p}_2 & \cdots & \vec{p}_m^T \cdot \vec{p}_m \end{pmatrix} \\
&= \begin{pmatrix} \vec{p}_1^T \cdot \vec{p}_1 & V^1_{p^1p^2} + V^2_{p^1p^2} & \cdots & V^1_{p^1p^m} + V^m_{p^1p^m} \\ V^1_{p^2p^1} + V^2_{p^2p^1} & \vec{p}_2^T \cdot \vec{p}_2 & \cdots & V^2_{p^2p^m} + V^m_{p^2p^m} \\ \vdots & & \ddots & \vdots \\ V^1_{p^mp^1} + V^m_{p^mp^1} & V^2_{p^mp^2} + V^m_{p^mp^2} & \cdots & \vec{p}_m^T \cdot \vec{p}_m \end{pmatrix} \\
&= \begin{pmatrix} \vec{p}_1^T \cdot \vec{p}_1 & V^1_{p^1p^2} & \cdots & V^1_{p^1p^m} \\ V^1_{p^2p^1} & 0 & \cdots & 0 \\ \vdots & \ddots & & \vdots \\ V^1_{p^mp^1} & 0 & \cdots & 0 \end{pmatrix} + \begin{pmatrix} 0 & V^2_{p^1p^2} & \cdots & 0 \\ V^2_{p^2p^1} & \vec{p}_2^T \cdot \vec{p}_2 & \cdots & V^2_{p^2p^m} \\ \vdots & \ddots & & \vdots \\ 0 & V^2_{p^2p^m} & \cdots & 0 \end{pmatrix} \\
&+ \cdots + \begin{pmatrix} 0 & \cdots & V^m_{p^1p^m} \\ 0 & \cdots & V^m_{p^2p^m} \\ \vdots & \ddots & \vdots \\ V^m_{p^1p^m} & \cdots & \vec{p}_m^T \cdot \vec{p}_m \end{pmatrix}.
\end{aligned}$$

We apply a similar strategy in our protocol for the computation of $X^T\vec{Y}$.

$$\begin{aligned}
X^T\vec{Y} &= \begin{pmatrix} p_1^1 & \cdots & p_n^1 \\ p_1^2 & \cdots & p_n^2 \\ \vdots & \ddots & \vdots \\ p_1^m & \cdots & p_n^m \end{pmatrix} \cdot \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \\
&= \begin{pmatrix} \vec{p}_1^T \cdot \vec{Y} \\ 0 \\ \vdots \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ \vec{p}_2^T \cdot \vec{Y} \\ \vdots \\ 0 \end{pmatrix} + \ldots + \begin{pmatrix} 0 \\ \\ \vdots \\ \vec{p}_m^T \cdot \vec{Y} \end{pmatrix}.
\end{aligned}$$

We now have a situation in which each party knows a vector and their sum is $X^T\vec{Y}$. Thus, Alice generates a vector $\vec{r}$ with n different random values, passes this $\vec{r}$ to the last party. Each party adds the vector given to its own vector and passes to the previously numbered party. When the vector is back to Alice, she subtracts $\vec{r}$ and adds her vector to obtain $X^T\vec{Y}$. This the the same as the SECURE SUM PROTOCOL.

Note that Alice does not know X^T; thus, knowledge of $X^T\vec{Y}$ will not enable it to derive the private values of $\vec{Y}$ when Alice is not the party supplying $\vec{Y}$ (even knowledge of $\vec{p}_1^T$ and $\vec{p}_1^T \cdot \vec{Y}$ does not reveal anything (unless $n = 1$, but we usually have more than one data point). Also, if $\vec{Y}$ is known by Alice and no other party, Alice cannot learn data from another party.

Hence Alice will get X^TX and $X^T\vec{Y}$. She can compute $\vec{\beta}$ by inverting X^TX and multiplying with $X^T\vec{Y}$. In the last step of the protocol, Alice broadcasts $\vec{\beta}$ to all parties.

We now introduce a protocol that distributes $\vec{\beta}$ in shares to at least two parties. This protocol works as before except that now we have some specific differences.

1. All parties will engage in the protocol for calculating X^TX and the output will go to Alice.

2. All parties and the party holding $\vec{Y}$ (say Yuri) will compute $X^T\vec{Y}$ and the output will go to a different party (than Alice). The easiest is for $X^T\vec{Y}$ to go to Yuri [4].

3. Alice and Yuri will multiply the matrix $A = (X^TX)^{-1}$ and the vector $\vec{B} = X^T\vec{Y}$ to obtain shares.

Step 1 and Step 2 are essentially as before. Step 3 can be achieved again by

[4]Note that Yuri is unable to find any private data from X^T, even though he has X^TY and Y, since this matrix-vector multipication carried out with the help SECURE SCALAR PRODUCT protocol.

using the SCALAR PRODUCT PROTOCOL WITH PRIVATE SHARES.

$$\begin{aligned} A\vec{B} &= \begin{pmatrix} (a_1^1, \cdots, a_m^1)^T \cdot \vec{B} \\ (a_1^2, \cdots, a_m^2)^T \cdot \vec{B} \\ \vdots \\ (a_1^m, \cdots, a_m^m)^T \cdot \vec{B} \end{pmatrix} \\ &= \begin{pmatrix} V^1_{\vec{a^1}\vec{b}} + V^2_{\vec{a^1}\vec{b}} \\ V^1_{\vec{a^2}\vec{b}} + V^2_{\vec{a^2}\vec{b}} \\ \vdots \\ V^1_{\vec{a^m}\vec{b}} + V^2_{\vec{a^m}\vec{b}} \end{pmatrix} = \begin{pmatrix} V^1_{\vec{a^1}\vec{b}} \\ V^1_{\vec{a^2}\vec{b}} \\ \vdots \\ V^1_{\vec{a^m}\vec{b}} \end{pmatrix} + \begin{pmatrix} V^2_{\vec{a^1}\vec{b}} \\ V^2_{\vec{a^2}\vec{b}} \\ \vdots \\ V^2_{\vec{a^m}\vec{b}} \end{pmatrix}. \end{aligned} \tag{5.4}$$

In this way, the output $\vec{\beta}$ will be shared between two parties. If the fact that X^TX is known to Alice is of some concern [70], a variant of the protocol where X^TX is discovered in distributed shares can be obtained if Alice and Bob skip the last matrix transmission in the protocol from the previous subsection. That is, Bob does not send his sum matrix to Alice. Thus, Alice will hold $A - R$ and Bob will hold $B + C + R$, which will serve as shares for the output. Similarly, rather than Yuri holding $X^T\vec{Y}$, the computation of the sum by passing a vector and accumulating can be halted before the last transmission. Then $(X^TX)^1X^T\vec{Y} = (A_1 + B_1)^{-1}(\vec{Z}_1 + \vec{Z}_2) = (A_1 + B_1)^{-1}\vec{Z}_1 + (A_1 + B_1)^{-1}\vec{Z}_2$. Multiplication and inversion of a matrix sum can be performed with dedicated protocols [38](pages 7-8).

5.1.3 Model Diagnosis

As we mentioned earlier, the calculation of $\vec{\beta}$ is an important step in regression, but it is only the first step. The other steps include diagnostics and model selection. Statistics reflecting the goodness of fit of a model include the correlation coefficient R^2 and the adjusted R^2. The residuals play an essential role in diagnostics. Once $\vec{\beta}$ is available, we can calculate the fitted or predicted responses as $\hat{\vec{Y}} = X\vec{\beta}$. The column vector of residues is $\hat{\varepsilon} = \vec{Y} - \hat{\vec{Y}}$ and the residual for the $i - th$ data point is $\hat{\varepsilon}_i = y_i - \hat{y}_i$. Then

$$R^2 = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y}_i)^2}. \tag{5.5}$$

For simplicity, we will assume that only 3 parties are involved. For more parties, we only need to extend the calculation of a sum of vectors among more parties by passing around an accumulator vector. If $\vec{\beta}$ has been calculated to make it available to all parties (the version without shares), then the data matrix X has columns owned by each party and

$$\begin{aligned}
X\vec{\beta} &= \begin{pmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ \vdots & & \\ a_n & b_n & c_n \end{pmatrix} \cdot \begin{pmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{pmatrix} \\
&= \begin{pmatrix} a_1\beta_1 \\ a_2\beta_1 \\ \vdots \\ a_n\beta_1 \end{pmatrix} + \begin{pmatrix} b_1\beta_2 \\ b_2\beta_2 \\ \vdots \\ b_n\beta_2 \end{pmatrix} + \begin{pmatrix} c_1\beta_3 \\ c_2\beta_3 \\ \vdots \\ c_n\beta_3 \end{pmatrix}. \qquad (5.6)
\end{aligned}$$

Again, this is a sum of vectors each known by one party, and it can be computed by summing and passing an accumulator initiated with random values by the first party.

When $\vec{\beta}$ is not publicly available, that is $\vec{\beta}$ is distributed by shares (Equation (5.4)), then privacy-preserving calculation is more challenging. Here we have

$$\begin{aligned}
& X\vec{\beta} \\
&= \begin{pmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ \vdots & & \\ a_n & b_n & c_n \end{pmatrix} \cdot \begin{pmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{pmatrix} \\
&= \begin{pmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ \vdots & & \\ a_n & b_3 & c_3 \end{pmatrix} \cdot \left[\begin{pmatrix} V^1_{\vec{p}^1\vec{\beta}} \\ V^1_{\vec{p}^2\vec{\beta}} \\ V^1_{\vec{p}^3\vec{\beta}} \end{pmatrix} + \begin{pmatrix} V^2_{\vec{p}^1\vec{\beta}} \\ V^2_{\vec{p}^2\vec{\beta}} \\ V^2_{\vec{p}^3\vec{\beta}} \end{pmatrix} \right] \\
&= \begin{pmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ \vdots & & \\ a_n & b_n & c_n \end{pmatrix} \cdot \begin{pmatrix} V^1_{\vec{p}^1\vec{\beta}} \\ V^1_{\vec{p}^2\vec{\beta}} \\ V^1_{\vec{p}^3\vec{\beta}} \end{pmatrix} + \begin{pmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ \vdots & & \\ a_n & b_n & c_n \end{pmatrix} \cdot \begin{pmatrix} V^2_{\vec{p}^1\vec{\beta}} \\ V^2_{\vec{p}^2\vec{\beta}} \\ V^2_{\vec{p}^3\vec{\beta}} \end{pmatrix}. \qquad (5.7)
\end{aligned}$$

Hence, we need a protocol for securely computing $X\vec{V}^1_{p\beta}$ and $X\vec{V}^2_{p\beta}$ where X is the vertically partitioned matrix and $\vec{V}^1_{p\beta}$ is a vector belonging to one of the parties. In our case we can assume it is Alice. The vector $X\vec{V}^2_{p\beta}$ belongs to Yuri, which may or may not have any values in X. If we can calculate the scalar product when one vector is vertically partitioned and the other one has all its entries known to one party, the computation of $X\vec{\beta}$ will again reduce to a sum of vectors distributed among the parties.

We are unaware of such protocol in the literature, so we propose here a solution based on Scalar Product Protocol with Private Shares.

The following protocol is for computing $\vec{p}^T \cdot \vec{y}$, where $\vec{p} = (p_1, \ldots, p_m)$, each entry p_i is known to the i-th party and the vector $\vec{y}$ is know to the first party [5].

Protocol 13 *1. The commodity server generates two random vectors* [6] $\vec{\Psi}$ *and* $\vec{\Pi}$ *of size* m, *and lets* $r_a + r_b = \vec{\Psi}^T \cdot \vec{\Pi}$, *where* r_a *(or* r_b*) is a randomly generated number. Then the server sends* $(\vec{\Psi}, r_a)$ *to the first party (lets say it is Alice). It send* r_b *to the second party (say Bob). It also sends* $\vec{\Pi}_i$ *to the i-th party.*

2. *Alice computes a perturbed version* $\hat{\vec{y}} = \vec{y} + \vec{\Psi}$ *of its vector and sends the i-th entry to the i-th party. Each party computes* $p_i\hat{y}_i = p_i y_i + p_i\vec{\Psi}_i$. *That is, the i-th party gets* $p_i(y_i + \vec{\Psi}_i)$.

3. *Each of the parties perturbs its value with the random number provided by the commodity server* $\hat{p}_i = p_i + \vec{\Pi}_i$ *and sends it to Alice. Thus, Alice obtains the vector* $\vec{p} + \vec{\Pi}$.

4. *The parties engage in a* secure sum protocol, *by which the first party passes* $p_1y_1 + p_1\vec{\Psi}_1$ *to the m-th party. The i-th party adds* $p_iy_i + p_i\vec{\Psi}_i$ *to the sum and passes it to the* $(i-1)$*-th party until the second party (Bob) has* $\vec{p}^T \cdot \vec{y} + \vec{p}^T \cdot \vec{\Psi}$.

5. *Bob generates a random number* V_2, *and computes* $\vec{p}^T \cdot \hat{\vec{y}} + (r_b - V_2)$. *He sends this result to Alice.*

[5]The case where the owner of $\vec{y}$ is not an owner of an entry p_i can be handled by this same protocol, but has even more relaxed privacy settings.

[6]All entries are random numbers.

6. *Alice adds* $r_a - \vec{\Psi}^T \cdot (\vec{p} + \vec{\Pi})$ *to the value from Bob and calls it* V_1. *This is* $V_1 = r_a - \vec{\Psi}^T \cdot (\vec{p} + \vec{\Pi}) + \vec{p}^T \cdot \hat{\vec{y}} + (r_b - V_2) = r_a + r_b - \vec{\Psi}^T \cdot \vec{\Pi} - \vec{\Psi}^T \cdot \vec{p} + \vec{p}^T \cdot (\vec{y} + \vec{\Psi}) - V_2 = \vec{p}^T \cdot \vec{y} - V_2$.

This protocol produces distributed shares V_1 for Alice and V_2 for Bob. The shares appear random to each but $V_1 + V_2 = \vec{p}^T \cdot \vec{y}$.

Thus, using this later scalar product protocol for the scalar product calculations in Equation (5.7), every party will get its shares and

$$\begin{aligned}\hat{\vec{Y}} = X\beta &= \begin{pmatrix} a_1 & b_1 & c_1 \\ \vdots & & \\ a_n & b_n & c_n \end{pmatrix} \cdot \begin{pmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{pmatrix} \\ &= \begin{pmatrix} V_1^1 \\ \vdots \\ V_n^1 \end{pmatrix} + \begin{pmatrix} V_1^2 \\ \vdots \\ V_n^2 \end{pmatrix} + \begin{pmatrix} V_1^3 \\ \vdots \\ V_n^3 \end{pmatrix} + \begin{pmatrix} V_1^4 \\ \vdots \\ V_n^4 \end{pmatrix}. \end{aligned} \tag{5.8}$$

It is clear now that by using a secure ADD VECTORS PROTOCOL we can obtain the column vector of residues $\hat{\varepsilon} = \vec{Y} - \hat{\vec{Y}}$. Using this, the coefficient R^2 can also be computed by the parties without any of them revealing their data.

5.1.4 Conclusion

We have presented practical algorithms for performing privacy-preserving regression in the more sensitive case, namely, where the response variable is private. Naturally, our methods apply as well when the response variable is public. Moreover, we have resolved both the case where we have two parties and the general case of more than two parties. Most importantly, we have addressed the second phase of the regression task, the model valuation phase. This last point is very important, as a poor model fit may indicate the need to repeat the first phase. Preserving privacy while performing both phases several times is crucial for the overall success of the regression task. If there were information leaks in either phase, iteration of the phases would increase the lack of privacy.

Privacy has a cost trade-off. Our algorithms are efficient because they offer only a constant overhead and are linear in the number of parties. The secure

Scalar Product Protocol with Private Shares performs $4n$ computations rather than n for two vectors of dimension n. All our protocols are based on the same number of scalar product operations and occasionally a secure sum protocol. This results in an overall complexity of $O(4mC(n))$ where $C(n)$ is the complexity on a consolidated database.

5.2 Clustering Algorithms

Clustering is defined as finding groups in the database *"by some natural criterion of similarity"* [40] or *"the objects are clustering or grouped based on the principle of maximizing the inter-class similarity and minimizing the intra-class similarity"* [57].

There are many algorithms for clustering [18], but they lie in some categorization like "Representative-based clustering" or "Density based clustering".

Representative-based clustering algorithms like K-means, K-medoids require representative points for clusters (e.g. center based on some metric) while density based clustering algorithms like *DBSCAN* require parameters for defining what is the minimum number of objects/points that could be set as a cluster. Moreover, clusters in K-means and K-medoids algorithms are always spherical. Non-convex clusters are detected with *DBSCAN*. The *DBSCAN* [42] algorithm almost fully satisfies the requirements mentioned above for applying on large databases.

A recent paper [101], which was published after our paper [7] on privacy-preserving *DBSCAN*, claims to produce the same results as outlined here and in our published paper. Their clustering algorithm mentions the use of R-Trees, but they are applied locally by each party. Obviously, this restricts the algorithm to the case of vertically partitioned data, since this is the case when parties could construct local R-Trees. However, these local R-Trees will not always coincide with the global R-Tree, and more importantly, local range queries retrieve a superset of a range query on the global R-Tree. This results in the generation of extra candidates in their algorithm. Thus, their approach is not only slower than ours, but could lead, by repeated usage of their Protocol 2 or their Protocol 3, to information leak. Moreover, because their protocols are not ideal, they reveal some information [7], the repeated invocation of range queries results in what we see as an unreasonable amount of information leak in the end.

5.2.1 Privacy-Preserving *DBSCAN*

DBSCAN's notion of cluster [42] is a region of high density[8]. *DBSCAN* has two parameters, ϵ (for the granularity of the histogram) and $MinPts$ (for the threshold on the height of the density). For *DBSCAN* those points that have high density around them are called core-points. *DBSCAN* is interested not only in collecting all core points, but also all points in $N_\epsilon(\vec{p})$, for all core points $\vec{p}$.

Definition 3 *We write* $N_\epsilon(\vec{p}) = \{\vec{q} \in \mathbb{D} \mid dist(\vec{p}, \vec{q}) \leq \epsilon\}$ *for the* ϵ-neighborhood of a point $\vec{p}$. *We say a point* $\vec{q}$ *is a* core point *if* $|N_\epsilon(\vec{q})| \geq MinPts$.

Lemma 2 *[42] Let* $\vec{p}$ *be core. Then, the set of all points* $\vec{q}$ *such that* $\exists\ \vec{p} = \vec{p}_0, \vec{p}_1, \ldots, \vec{p}_n = \vec{q}$, *so that* $\vec{p}_i \in N_\epsilon(\vec{p}_{i-1})$ *and* $|N_\epsilon(\vec{p}_{i-1})| \geq MinPts$, *(for* $i = 1, \ldots, n$*) is a cluster. If* C *is a cluster and* $\vec{p} \in C$ *is a core point, then* C *equals all the points* $\vec{q}$ *such that* $\vec{p} = \vec{p}_0, \vec{p}_1, \ldots, \vec{p}_n = \vec{q}$ *with* $\vec{p}_i \in N_\epsilon(\vec{p}_{i-1})$ *and* $|N_\epsilon(\vec{p}_{i-1})| \geq MinPts$, *for* $i = 1, \ldots, n$.

DBSCAN finds all clusters defined by Lemma 2 [42]. Therefore, *DBSCAN* works essentially by finding a core point and then all points that can be reached by a sequence of ϵ-neighborhoods [42]. It starts with an arbitrary point $\vec{p}$ and checks whether $\vec{p}$ is a core point. If it is, $\vec{p}$ becomes $\vec{p}_0$ and then all points in $N_\epsilon(\vec{p}_0)$ are examined to see if they are core points (while $N_\epsilon(\vec{p}_0)$ is added to the cluster). All points $\vec{p}_i$ found to have $N_\epsilon(\vec{p}_i) \geq MinPts$ are included to the cluster as core points, and also their ϵ-neighbors, until no more points can be added. Thus, the fundamental operation in *DBSCAN* is "Given a point $\vec{p} \in \mathbb{D}$, retrieve all points $\vec{q} \in N_\epsilon(\vec{p})$". From the central operation, the following operations are readily

[7]For instance, their use of Yao comparisons reveals information about other parties' distance data.

[8]In a statistical sense this is a loose notion of what is a cluster, as those areas in a histogram with density above a certain threshold, but it illustrates the flavor of *DBSCAN* as it is closer to descriptive statistics than parametric inference.

	DBSCAN **on** *DNSP*	**PP** *DBSCAN*
Privacy	None	Under semi-homest model
Performance	Communication $\Theta(N)$ Very close to the size of the database Computation $\Theta(N \log N)$ Essentialy building an R-Tree	A constant factor of overhead

Table 5.1: Comparison of the *DBSCAN* in *DNSP* with privacy-preserving *DBSCAN*.

implemented.

• OP1 (**Is $\vec{p}$ core?* *): Given a point $\vec{p}$ in the database $\mathbb{D}$, is $|N_\epsilon(\vec{p})| \geq MinPts$?

• OP2 (**Find neighborhood of core* *): Given a point $\vec{p}$ in the database $\mathbb{D}$ with $|N_\epsilon(\vec{p})| \geq MinPts$, retrieve all other points $\vec{q} \in N_\epsilon(\vec{p})$.

• OP3 (**Core neighbors of core* *): Given a core point $\vec{q}$, retrieve a list of core points in $N_\epsilon(\vec{q}) \setminus \{\vec{q}\}$?[9]

The fundamental operation can be performed efficiently using data structures for associative queries, like *KD*-Trees, *R*-Trees, *SASH* or other multidimensional access methods [53]. Obviously, a privacy-preserving version is now achieved using our data structures from the previous chapter.

[9]While this operation is technically different to OP2 or OP1 and this is useful for the correctness of the pseudocode, in practical implementations of the pseudocode, when OP3 follows an OP2 or and OP1, the work of the previous operation would be used for a fast implementation of OP3.

Below we present a pseudocode for *DBSCAN* algorithm that uses OP1, OP2 and OP3.

```
• $S \leftarrow D; i \leftarrow 0;$
• WHILE there is a core point $p \in S$ (“use OP1”) DO
    – $S \leftarrow S \setminus \{p\};$
    – $p_0 \leftarrow p;$
    – $C_i \leftarrow N_{Eps}(p_0);$
    – $CoreList \leftarrow$ OP3($p_0$); (“a list of core points in $N_{Eps}(p_0)$”)
    – IF $CoreList == \emptyset$; THEN
        ∗ $S \leftarrow S \setminus C_i;$
        ∗ OUTPUT i-th cluster $C_i$;
        ∗ $i++;$
    – ELSE
        ∗ $p_j \leftarrow$ FIRST($CoreList$);
        ∗ $CoreList \leftarrow CoreList \setminus p_j;$
        ∗ $C_i \leftarrow C_i \cup N_{Eps}(p_j)$; (“use OP2”)
        ∗ $CoreList \leftarrow$ APPEND($CoreList$,OP3($N_{Eps}(p_j)$) $\setminus$ $CoreList$);
```

Chapter 6

Final Remarks

6.1 Summary of Contributions

Now let us summarise what was achieved in this thesis. We do this by contrasting with the aims outlined in the introduction.

- Extend and migrate existing clustering algorithms to the privacy-preserving data mining context.

  The widely used clustering algorithm *DBSCAN* was presented in the privacy context.

- Develop more general tools in Privacy-Preserving Data Mining , in particular, rather than focus on each individual clustering algorithm, we revise the general data structures used in many algorithms.

  Data structures are essential in almost every information retrieval when working with big databases, which is usually the case in data mining. Thus, we presented here three data structures: the *R*-Trees, *KD*-Trees and the *SASH*. The *SASH* is relatively new, but as was shown in the relevant literature is also very successful, especially for *k*-NN queries. These data structures will not only enable efficient information retrieval, but they

will alleviate the implementation/adaptation issues for different kinds of data mining algorithms that use these data structures.

- Develop general secure multiparty computational tools that will help algorithms that are not designed for PPDM systematically to be transferred to the PPDM context.

  Several new algorithms were presented here for secure-multiparty computations. The protocols for comparison (Yao's millionaires problem), the NEW SCALAR PRODUCT PROTOCOL and FINDING THE MAXIMUM VALUE IN THE SUM OF VECTORS are fundamental building blocks for many privacy-preserving algorithms. PPDM literature shows the essential role they play in this area. One should not underestimate the importance of privacy-preserving metrics as well, which we have integrated into the privacy context.

In particular, in Chapter 2 the main contributions are

- Protocol for the MAXIMUM VALUE IN THE SUM OF VECTORS.
- New Scalar product protocol.
- New solution for Yao's millionaires problem with private shares.
- NEW DIVISION PROTOCOL WITH PRIVATE SHARES

Task	**Performance**	
	Communication	**Computation**
Comparison ($a \leq b?$)	$3\log_2(a+b)+2r$, where r is a 2 bit number	Generation of 3 random numbers and 4 basic operations
Scalar product	$\Theta(n)$	$\Theta(n)$
Division	Three scalar products of two vectors with size 2	Three scalar products of two vectors with size 2 and generation of two random numbers

Table 6.1: Performace of basic SMC protocols.

In Chapter 3 we discussed metrics and k-near neighbour queries in the privacy context in different cases of partitioning. We presented new algorithms for the following tasks.

- k-near neighbours in horizontal and vertically partitioned data, where we compared information disclosure with the ideal case, and presented an alternative secure solution with quadratic performance.
- Metrics computations that consider the case when we need output distributed by shares. In particular our solution for a secure chessboard metric for vertically partitioned data enabled by solution for Yao's millionaires problem with more that two parties.
- The possibility of combining the different local metrics into global metrics.

Task	**Performance**	
	Communication	**Computation**
Horizontally partitioned data		
Euclidian	$\Theta(n)$	One scalar product of vectors with size n operations
Chessboard	$\Theta(n)$	Generation of one permutation and finding maximum within n numbers
Cosine	$\Theta(n)$	Three scalar product of vectors with size n and two multiplications
Vertically partitioned data		
Minkowski	$\Theta(n)$	$\Theta(n)$
Chessboard	$\Theta(p^2)$, where p is the number of parties	$\Theta(p^2 n)$
Cosine	$\Theta(n)$	Four scalar products of vectors with size n and one division protocol

Table 6.2: Performace of SMC metrics.

The data structures chapter provides privacy-preserving versions of *KD*-Trees, *R*-Trees and the *SASH*. We have presented experimental results for the *SASH* compared with the Fagin's A0 algorithm, which without any doubt show the great efficiency in the favour of the *SASH*. While Fagin's A0 algorithm has performance very close to the size of the entire database, the *SASH* has a performance well below inspecting the entire database, and close to logarithmic. Having privacy-preserving data structures will enable systematic adaptation of clustering algorithms, which are based on associative queries, into the privacy context. An example of this is the adaptation for the semi-honest model of the well known clustering algorithm *DBSCAN* in Chapter 5. In the same chapter we introduced methods for carrying out statistical analysis such as regression while preserving privacy. This is particularly interesting, because we developed a privacy-preserving tool for model diagnosis.

6.2 New Avenues for Research

Some of our protocols require the parties to share explicit knowledge of the topology of the data structure. Other researchers have also dealt with this situation. For example, Du and Zhan, have a protocol where parties share the topology of a decision tree [39]. Could this be actually kept secret and away from all the parties, but still process associative queries efficiently ? In a sense, we have seen that it is easy for parties to have secret shares of a Boolean value like the result of a Yao comparison. But these are secret shares of one bit, and it should be theoretically possible to have secret shares of other values encoded as bit strings, among these the pointer and the information fields of nodes in data structures. We illustrate this discussion now, and show that it opens an interesting line of investigation.

Let us assume there are given 9 points $G(a_1, b_1)$, $F(a_2, b_2)$, $H(a_3, b_3)$, $C(a_4, b_4)$, $A(a_5, b_5)$, $D(a_6, b_6)$, $J(a_7, b_7)$, $I(a_8, b_8)$ and $K(a_9, b_9)$, where the data is vertically partitioned between Alice and Bob. They have constructed the following binary tree (see Figure 6.1) according to some distance function *dist* [1].

[1]This could return the order of the letters in the alphabet, which is used for labelling the points.

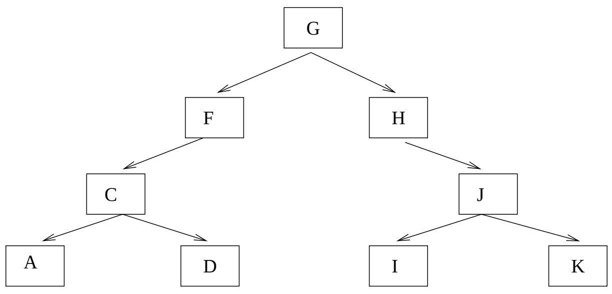


Figure 6.1: Example of a binary search tree.

The tree is distributed with secret shares in the following way:

For each point, for instance G, Alice has ID_a and Bob has ID_b, such that $ID_a + ID_b = ID(G) = 1$. Moreover, again for every node , for instance G, there is $nextNode(ID(G), dir_a, dir_b)$ function, where

$$dir_a + dir_b = \begin{cases} 0 & \text{left,} \\ 1 & \text{right,} \end{cases}$$

which outputs the ID of the next node in shares, depending on the direction, and $NULL$ if it is a leaf node. For example, if $dir_a + dir_b = 1$ (right), then $nextNode(ID(G), dir_a, dir_b)$ will send ID_a to Alice and ID_b to Bob, such that $ID_a + ID_b = ID(H) = 3$.

They have also $rootID_a$ and $rootID_b$, where $rootID_a + rootID_b = ID(root)$, which is in this case is the same as $ID(G)$.

Let us now see, how we can insert a new point into this tree. First we have to find the place to insert. For this, we need to be able to navigate through the tree. Now, a new point $B(a_{10}, b_{10})$ should be inserted. Assuming the existence of a $getDir$ function, the following algorithm could be applied.

1. The function $getDir(rootID_a, rootID_b, ID_a(B), ID_b(B))$ compares the root (point G) with B, by using $dist$, and outputs dir_a to Alice and dir_b to Bob. Note that in this case $dir_a + dir_b = 0$ (left).

2. They apply the function $nextNode(rootID_a, rootID_b, dir_a, dir_b)$ which provides Alice with $ID_a(F)$ and Bob with $ID_b(F)$.

3. Again the function $getDir(ID_a(F), ID_b(F), ID_a(B), ID_b(B))$ will provide Alice dir_a and Bob dir_b. Here again $dir_a + dir_b = 0$ (left).

4. The $nextNode(ID_a(F), ID_b(F), dir_a, dir_b)$ will send Alice $ID_a(C)$ and Bob with $ID_b(C)$.

5. Repeating step 2 and step 3 until node A, the $nextNode$ function will point to $NULL$, which means the point B should be inserted here. The tree should be updated as well, that is $nextNode(ID_a(A), ID_b(A), dir_a, dir_b)$ where $dir_a + dir_b = 1$ (right) should now point to the node B, namely the output should be $ID_a(B)$ and $ID_b(B)$.

This example demonstrates that it is possible to navigate through the binary search tree without even Alice or Bob knowing where they are going, or which direction they have taken. Everything is kept in secret shares.

Hence, our further research will concentrate on these issues. In particular, we are interested in several topics.

- How to construct such a tree. Investigate possibilities of constructing different trees satisfying these requirements.
- Construct protocols for the actual functions $getDir$ and $nextNode$ for every tree.
- Analyse the possible information leakage.
- Perform experiments for different data sets.

Bibliography

[1] Private lives; public records — in today's networked world is personal information too easy to access? *Computerworld*, 31(6):88–90, September 1997.

[2] Get the insight you need to achieve business goals, Retrieved July 13, 2007. http://www.spss.com/clementine/index.htm.

[3] R. Adderley. Report in kdnuggets news, 2002.

[4] D. Agrawal and C. Aggarwal. On the design and quantification of privacy preserving data mining algorithms. In *Proceedings of the 20th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Santa Barbara, CA, May 21-23 2001. ACM Press.

[5] R. Agrawal, T. Imielinski, , and A. Swani. Database mining: a performance perspective. *IEEE Trans. of Knowledge and data engineering*, 5(6):914–925, 1993. Special issue on learning and discovery in knowledge-based databases.

[6] R. Agrawal and R. Srikant. Privacy-preserving data mining. In W. Chen, J. Naughton, and P. A. Bernstein, editors, *Proceedings of SIGMOD*, pages 439–450, 2000.

[7] A. Amirbekyan and V. Estivill-Castro. Privacy preserving *DBSCAN* for vertically partitioned data. In *IEEE International Conference on Intelligence and Security Informatics, ISI 2006*, volume 3975, pages 141–153, San Diego, CA, USA, May 23-24 2006. Springer Verlag, Lecture Notes in Computer Science.

[8] A. Amirbekyan and V. Estivill-Castro. The privacy of k-nn retrieval for horizontal partitioned data — new methods and applications. In J. Bailey and A. Fekete, editors, *Eighteenth Australasian Database Conference (ADC2007)*, volume 63 of *Conferences in Research and Practice in Information Technology (CRPIT)*, pages 33–42, Ballarat, Victoria, Australia, January 2007. CORE, Australasian Computer Society.

[9] M. Ankerst, G. Kastenmueller, H. Kriegel, and T. Seidl. Nearest neighbor classification in 3D protein databases. In *Proc. 7th Int. Conf. on Intelligent Systems for Molecular Biology (ISMB-99)*, pages 34–43, 1999.

[10] M. Ankerst, H. Kriegel, and T. Seidl. A multi-step approach for shape similarity in image databases. *IEEE Transactions on Data Engineering*, 10(6):996–1004, 1998.

[11] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching. In *5th ACM-SIAM Symposium on Discrete Algorithms*, pages 573–582, 1994.

[12] M. Atallah, M. Bykova, J. Li, K. Frikken, and M. Topkara. Private collaborative forecasting and benchmarking. In *proceedings of ACM workshop on Privacy in the electronic society (WPES)*, pages 103–114, 2004.

[13] M.J. Atallah and W. Du. Secure multi-party computational geometry. In *Proceedings of the 7th International Workshop on Algorithms and Data Structures*, volume 2125, pages 165–179. Springer Verlag, Lecture Notes in Computer Science, August 8-10 2000.

[14] S. Avidan and M. Butman. Blind vision. In S. Leonardis, H. Bischof, and A. Pinz, editors, *Computer Vision - ECCV 2006, 9th European Conference on Computer Vision, Part III*, volume 3953, pages 1–13, Graz, Austria, May 7-13 2006. Springer Verlag, Lecture Notes in Computer Science.

[15] D. Beaver. Server-assisted cryptography. In *NSPW '98: Proceedings of the 1998 workshop on new security paradigms*, pages 92–106, New York, NY, USA, 1998. ACM Press, ISBN: 1-58113-168-2.

[16] J.L. Bentley. Multidimensional binary search trees used for associative retrieval. *Communications of the ACM*, 18(9):509–517, 1975.

[17] S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-tree: An index structure for higher dimensional data. In *Proceedings of the 22nd Conference on Very Large Data Bases (VLDB)*, pages 28–39, 1996.

[18] P. Berkhin. Survey of clustering data mining techniques. Technical report, Accrue Software, San Jose, CA, 2002. Available from: http://citeseer.nj.nec.com/berkhin02survey.html.

[19] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is nearest neighbor meaningful ? In *International Conference on Database Theory*, pages 217–225, Jerusalem, Israel, Janunary 1999.

[20] L. Brankovic and V. Estivill-Castro. Privacy issues in knowledge discovery and data mining. In C.R. Simpson, editor, *AICEC99 Conference Proceedings*, pages 89–99, Melbourne, Australia, July 14-16 1999. Swinburne University of Technology, Australian Institute of Computer Ethics.

[21] A.J. Broder. Data mining, the Internet and privacy. In B. Masand and M. Spiliopoulou, editors, *Proceedings of WEBKDD-99, International Workshop on Web usage Analysis and user Profiling*, pages 56–73. Springer-Verlag Lecture Notes in Artificial Intelligence 1836, 2000.

[22] C. Cachin. Efficient private bidding and auctions with an oblivious third party. In *Proceedings of the 6th ACM Conference on Computer and Communications Security*, pages 120–127, Singapore, November 1999. SIGSAC, ACM Press.

[23] J. Catlett. Among those dark electronic mills: Privacy and data mining. In R. Ramakrishnan, editor, *ACM SIGKDD International Conference on knowledge discovery and data mining;*, page 4, Boston, MA, August 2002. ACM, Association for Computing Machinery.

[24] B. Chazelle and H. Edelsbrunner. An improved algorithm for constructing k-th order voronoi diagrams. *IEEE Transactions of Computers*, C(36):1349–1354, 1987.

[25] X. Cheng, R. Dolin, M. Neary, S. Prabhakar, K. Ravikanth, D. Wu, D. Agrawal, E. El Abbadi, M. Freeston, A. Singh, T. Smith, and J. Su. Scalable access within the context of digital libraries. In *Proceedings of the*

International Conference on Advances in Digital Libraries, pages 70–81, Washington, D.C., 1997.

[26] P. Ciaccia and M. Patella. PAC nearest neighbor queries: Approximate and controlled search in high-dimensional and metric spaces. In *Proceedings of the International Conference on Data Engineering*, pages 244–255, San Jose, California, March 2000.

[27] R. Clarke. Information technology and dataveillance. *Communications of the ACM*, 31(5):498–512, May 1988.

[28] R. Clarke. Introduction to dataveillance and information privacy, and definitions of terms. http://www.anu.edu.au/people/Roger.Clarke/DV/Intro.html, August 1997.

[29] R. Clarke. Person-location and person-tracking: Technologies, risks and policy implications. Introduction to Dataveillance and Information Privacy, and Definitions of Terms, October 1999. www.anu.edu.au/people/Roger.Clarke.

[30] C. Clifton. Protecting against data mining through samples. In *Thirteenth Annual IFIP WG 11.3 Working Conference on Database Security*, volume 173, pages 193–207, Seattle, WA, July 1999. Kluwer, B. V.

[31] C. Clifton. Using sample size to limit exposure to data mining. *Journal of Computer Security*, 8(4):281–307, 2000. Expanded version of IFIP-99 paper.

[32] C. Clifton and D. Marks. Security and privacy implications of data mining. In *SIGMOD Workshop on Data Mining and Knowledge Discovery*, Montreal, Canada, June 1996. ACM.

[33] G. Cunto, W. Lau and P Flajolet. An analysis of kd-trees: Kd-trees improved by local rotations. In F. Dehne, J.R. Sack, and N. Santoro, editors, *Algorithms and Data Structures, WADS-89*, volume 38295, pages 24–38, Ottawa, Canada, 1989. Springer-Verlag Lecture Notes in Computer Science.

[34] E. Davis (Chief Architect Intel Corp). Tera tera tera, presentation, slide 3, Retrieved July 6 2007. http://bt.pa.msu.edu/TM/BocaRaton2006/talks/davis.pdf.

[35] I. Dinur and K. Nissim. Revealing information while preserving privacy. In *Proceedings of the Twenty-Second ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 9-12, 2003, San Diego, CA, USA*, pages 202–210, 2003, ISBN: 1-58113-670-6.

[36] W. Du and M.J. Atallah. Privacy-preserving cooperative statistical analysis. In *Proceedings of the 17th Annual Computer Security Applications Conference (ACSAC)*, pages 102–110, New Orleans, Louisiana, December 10-14 2001. ACM SIGSAC, IEEE Computer Society.

[37] W. Du and M.J. Atallah. Secure multi-party computation problems and their applications: A review and open problems. In *Proceedings of the New Security Paradigms Workshop*, pages 13–22, Cloudcroft, New Mexico, USA, September 11-13 2001. SIGSAC, ACM Press.

[38] W. Du, Y.-S. Han, and S. Chen. Privacy-preserving multivariate statistical analysis: Linear regression and classification. In Berry M. W., U. Dayal, C. Kamath, and D.B. Skillicorn, editors, *2004 SIAM International Conference on Data Mining*, Lak Buena Vista, Florida, April 22nd-24th 2004.

[39] W. Du and Z. Zhan. Building decision tree classifier on private data. In V. Estivill-Castro and C. Clifton, editors, *Privacy, Security and Data Mining*, pages 1–8, Sydney, Australia, December 2002. IEEE ICDM Workshop Proceedings, Volume 14 in the Conferences in Research and Practice in Information Technology Series, Australian Computer Society.

[40] R. Duda and P. Hart. *Pattern Classification and Scene Analysis.* John Wiley & Sons, New York, NY, 1973.

[41] A. Einsenberg. Data mining and privacy invasion on the net. *Scientific American*, 274(3):120, 1996.

[42] M. Ester, H.P. Kriegel, S. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In

E. Simoudis, J. Han, and U. Fayyad, editors, *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD-96)*, pages 226–231, Menlo Park, CA, 1996. AAAI, AAAI Press.

[43] V. Estivill-Castro. Private representative-based clustering for vertically partitioned data. In R. Baeza-Yates, J.L. Marroquin, and E. Chávez, editors, *Fifth Mexican International Conference on Computer science (ENC 04)*, pages 160–167, Colima, Mexico, September 2004. SMCC, IEEE Computer Society Press.

[44] V. Estivill-Castro and L. Brankovic. Data swapping: Balancing privacy against precision in mining for logic rules. In *Data Warehousing and Knowledge Discovery, First International Conference*, volume 1676, pages 389–398. Springer Verlag, Lecture Notes in Computer Science, 1999.

[45] V. Estivill-Castro, L. Brankovik, and D.L. Dowe. Privacy in data mining. *Privacy - Law & Policy Reporter*, 6(3):33–35, September 1999.

[46] V. Estivill-Castro and J. Yang. Clustering Web visitors by fast, robust and convergent algorithms. *Int. J. Found. Comput. Sci.*, 13(4):497–520, 2002.

[47] A. Evfimievski, R. Srikant, R. Agrawal, and J. Gehrke. Privacy preserving mining of association rules. In O.R. Zaïne, editor, *8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 217–218, Edmonton, Alberta, Canada, July 23-26 2002. ACM Press.

[48] R Fagin. Combining fuzzy information from multiple systems. In *Proceedings of the Fifteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 216–226, Montreal, Canada, June 3-5 1996. ACM Press, ISBN: 0-89791-781-2.

[49] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *Proceedings ACM SIGMOD International Conference on Management of Data*, pages 419–429, Minneapolis, May 1994.

[50] U.M. Fayyad, G. Piatesky-Shapiro, and P. Smyth. From data mining to knowledge discovery: An overview. In U.M. Fayyad, G. Piatesky-Shapiro,

P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 1–36, Menlo Park, CA, 1996. AAAI Press / MIT Press.

[51] H. Ferhatosmanoglu, E. Tuncel, D. Agrawal, and A. El Abbadi. Approximate nearest neighbor searching in multimedia databases. In *7th IEEE International Conference on Data Engineering (ICDE)*, pages 503–511, Germany, April 2002.

[52] W. J. Frawley, G. Piatetsky-Shapiro, and C. J. Matheus. Knowledge discovery in databases: An overview. *AI Magazine*, 13(3):57–70, 1992.

[53] V. Gaede and O. Günther. Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231, June 1998.

[54] O. Goldreich. *The Foundations of Cryptography*, volume 2, chapter General Cryptographic Protocols. Cambridge University Press, 2004.

[55] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game (extended abstract). In A.V. Aho, editor, *Proceedings of the 19th ACM Annual Symposium on Theory of Computing*, pages 218–229, New York, 1987. ACM Press.

[56] A. Guttmann. R-trees: a dynamic index structure for spatial searching. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 47–57, 1984.

[57] J. Han and M Kamber. *Data Mining: Concepts and Techniques.* Morgan Kaufmann, 2000, ISBN: 1-55860-489-8.

[58] J. Hartigan. *Clustering Algorithms.* John Wiley & Sons, New York, NY, 1975.

[59] M.E. Houle. Navigating massive data sets via local clustering. In L. Getoor, T.E. Senator, P. Domingos, and C. Faloutsos, editors, *9th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 547–552, Washington, DC, 2003. ACM Press.

[60] M.E. Houle. SASH: a spatial approximation sample hierarchy for similarity. Technical Report RT-0517, IBM Tokyo Research Laboratory, March 5th 2003. 16 pages.

[61] M.E. Houle and J. Sakuma. Fast approximate similarity search in extremely high-dimensional data sets. In *21st International Conference on Data Engineering ICDE*, pages 619–630, Tokyo, Japan, April 5-8 2005. IEEE Computer Society.

[62] Z. Huang, W. Du, and B. Chen. Deriving private information from randomized data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA*, pages 37–48, 2005, ISBN: 1-59593-060-4.

[63] I. Ioannidis and A. Grama. An efficient protocol for Yao's millionaires' problem. In *36th Hawaii International Conference on System Sciences HICSS*, page 205, Big Island, HI, January 6th-9th 2003. IEEE Computer Society.

[64] I. Ioannidis, A. Grama, and M. J. Atallah. A secure protocol for computing dot-products in clustered and distributed environments. In *31st International Conference on Parallel Processing (ICPP)*, pages 379–384, Vancouver, BC, Canada, 20-23 August 2002. ISBN: 0-7695-1677-7.

[65] T. Kahveci and A. K. Singh. An efficient index structure for string databases. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB)*, pages 351–360, Rome, Italy, September 2001.

[66] M. Kantarcioglu and C. Clifton. Privacy-preserving distributed mining of association rules on horizontally partitioned data. In *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, Madison, Wisconsin, USA*, 2002.

[67] M. Kantarcioğlu and C. Clifton. Privately computing a distributed k-nn classifier. In J.-F. Boulicaut, editor, *8-th European Conference on Principles and Practice of Knowledge Discovery in Databases PKDD*, volume 3202, pages 279–290. Springer Verlag, Lecture Notes in Computer Science, 2004.

[68] H. Kargupta, S. Datta, Q. Wang, and K. Sivakumar. On the privacy preserving properties of random data perturbation techniques. In *Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM 2003),*

19-22 December 2003, Melbourne, Florida, USA, pages 99–106. IEEE Computer Society, ISBN: 0-7695-1978-4.

[69] A. F. Karr, X. Lin, A. P. Sanil, and J.P. Reiter. Secure regression on distributed databases. *Journal of Computational and Graphical Studies*, 14(2):1–18, 2005.

[70] A. F. Karr, X. Lin, A. P. Sanil, and J.P. Reiter. Secure statistical analysis of distributed databases. In A. G. Wilson, G. D. Wilson, and D.H. Olwell, editors, *Statistical Methods in Counterterrorism — Game Theory, Modeling, Syndromic Surveillance, and Biometric Authentication*, pages 237–261. Springer, 2006.

[71] N. Katayama and S. Satoh. The SR-tree: an index structure for high-dimensional nearest neighbour queries. In *ACM SIGMOD Conf. on Management of Data*, pages 369–380, Tucson, USA, 1997.

[72] L. Kaufman and P. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis.* John Wiley and Sons, New York, NY, 1990.

[73] V. Klikenborg. Trying to measure the amount of information that humans create, November 12, 2003. New York Times.

[74] F. Korn, N. Sidropoulos, C. Faloutsos, E. Siegel, and Z. Proropapas. Fast nearest neighbor search in medical image databases. In *Proceedings of the 22nd Internatial Conference on Very Lage Data Bases (VLDB)*, pages 215 – 226, Mumbai, India, 1996.

[75] K. C. Laudon. Markets and privacy. *Communications of the ACM*, 39(9):92–104, 1996.

[76] D.T. Lee. On k-nearest neighbors voronoi diagrams in the plane. *IEEE Transactions of Computers*, C(31):478–487, 1982.

[77] Y Lindell and Pinkas B. Privacy preserving data mining. In M. Bellare, editor, *Proceedings of CRYPTO-00 Advances in Cyptology*, volume 1880, pages 36–54, Santa Barbara, California, USA, August 20-24 2000. Springer-Verlag Lecture Notes in Computer Science.

[78] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay — a secure two-party compuatation system. In *Proceedings of USENIX security Symposium*, pages 287–302, 2004.

[79] C.J. Matheus, P.K. Chan, and G. Piatetsky-Shapiro. Systems for knowledge discovery in databases. *IEEE Transactions on Knowledge and Data Engineering*, 5:903–913, 1993.

[80] J. Mena. *Investigative Data Mining for Security and Criminal Detection.* Butterworth-Heinemann, US, 2003, ISBN: 0-07506-7613-2.

[81] K.S. Nash. Electronic profiling — critics fear systems may trample civil rights. *Computerworld*, 32(6):1,28, February 1998.

[82] Regents of the University of California. How much information?, Retrieved July 6, 2007. http://www2.sims.berkeley.edu/research/projects/how-much-info-2003/execsum.htm.

[83] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, volume 1592, pages 223–238. Springer Verlag, Lecture Notes in Computer Science, 1999.

[84] M. Perton. Emc rolls out USD 4 million petabyte array, Retrieved July 5, 2007. http://www.engadget.com/2006/01/30/emc-rolls-out-4-million-petabyte-array.

[85] N. Roussopoulos, S. Kelly, and F. Vincent. Nearest neighbor queries. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 71–79, San Jose, California, May 1995. ACM Press.

[86] A. P. Sanil, A. F. Karr, X. Lin, and J.P. Reiter. Privacy preserving regression modelling via distributed computation. In W. Kim, R. Kohavi, J. Gehrke, and W. DuMouchel, editors, *Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 677–682, Seattle, Washington, August 22nd-25th 2004. ACM.

[87] M. Shaneck, Y. Kim, and V. Kumar. Privacy preserving nearest neighbor search. Technical report, 2006.

[88] L. Shoundong, D. Yiqi, and Y. Qiyou. Secure multi-party computation solution to yao's millionares problem based on set-inclusion. In *Progress in Natural Science*, volume 15:9, pages 851–856, 2005.

[89] M.-C. Silaghi. Secure multi-party computation for selecting a solution according to a uniform distribution over all solutions of a general combinatorial problem. Cryptology ePrint Archive, Report 2004/333, 2004.

[90] V. Subrahmanian. *Principles of mutimedia database systems.* Morgan Kaufmann Publishers, Inc., San Francisco, CA, USA, 1999.

[91] L. Sweeney. k-anonymity: a model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(5):557–570, October 2002.

[92] J. Vaidya and C. C. Clifton. Privacy preserving association rule mining in vertically partitioned data. In *The Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 639–644, Edmonton, Canada, July 23-26 2002. SIGKDD, ACM Press.

[93] J. Vaidya and C. C. Clifton. Privacy-preserving k-means clustering over vertically partitioned data. In *Proceedings of the SIGKDD-ACM Conference of Data Mining*, pages 206–215, Washington, D.C., US, August 24-27 2003. ACM Press.

[94] J. Vaidya and C. Clifton. Privacy-preserving outlier detection. In *4th IEEE International Conference on Data Mining (ICDM 2004)*, Brighton, UK, November 2004. IEEE Computer Society, ISBN: 0-7695-2142-8.

[95] J. Vaidya and C. Clifton. Privacy-preserving top-k queries. In *21st International Conference on Data Engineering, ICDE 2005*, pages 545–546, Tokyo, April 2005. IEEE Computer Society.

[96] J. Vaidya, C. Clifton, and M. Zhu. *Privacy Preserving Data Mining*, volume 19 of *Advances in Information Security.* Springer, New York, NY, USA, 2006, ISBN: 978-0-387-25886-7.

[97] P. van der Putten and M. van Someren. CoIL challenge 2000: The insurance company case. Technical Report 2000-09, Leiden Institute of Advanced Computer Science, Amsterdam, June 22 2000.

[98] Seifert J. W. Data mining and homeland security: An overview, Retrieved July 13, 2007. www.fas.org/sgp/crs/intel/RL31798.pdf.

[99] S. M. Weiss and C. A. Kulikowski. *Computer Systems That Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning and Expert Systems*. Morgan Kaufmann, 1990, ISBN: 1-55860-065-5.

[100] I. Witten and E. Frank. *Data Mining — Practical Machine Learning Tools and Technologies with JAVA implementations*. Morgan Kaufmann Publishers, Inc., 2000.

[101] W. Xu, L. Huang, Y. Luo, Y. Yao, and W. Jing. Privacy-preserving *DBSCAN* clustering over vertically partitioned data. In *2007 International Conference on Multimedia and Ubiquitous Engineering (MUE 2007), 26-28 April, Seoul, Korea*, pages 850–856. IEEE Computer Society, 2007, ISBN: 978-0-7695-2777-2.

[102] A.C. Yao. Protocols for secure computation. In *IEEE Symposium of Foundations of Computer Science*, pages 160–164. IEEE Computer Society, 1982.

[103] B. Yu, C. Ooi, K.L. Tan, and H. Jagadish. Indexing the distance: an efficient method to KNN processing. In *Proccedings of the 27th International Conference on Very Large Data Bases (VLDB)*, pages 421–430, Rome, Italy, 2001.