Privacy-preserving Data Mining

Konstantin Tretyakov

October 17, 2005

Abstract

The relatively young subject of *privacy-preserving data mining* attempts to provide ways of executing complex statistical analyses on large databases without the need to fully disclose the contents of the databases themselves. There are two approaches that achieve this goal: one is based on cryptographical (and thus mostly number-theoretical) techniques; another uses statistical randomization methods. This paper aims to give a brief overview of the two methods. The sketch of the privacy-preserving version of the decision-tree construction algorithm ID3 is presented as an illustration of the cryptographic technique.

1 Introduction

It is well known that statistical analysis of large datasets can provide significant insights and guide to successful business decisions. A wide range of sophisticated algorithms have been developed during the recent decades, that allow to extract useful knowledge from data. Collectively these algorithms are known as *data-mining methods*.

A typical data-mining algorithm usually requires all data to be collected into a single warehouse before being analyzed. However, privacy and security considerations might often prevent this approach. For example, medical institutions might want to use data mining to identify trends and patterns in disease outbreaks. They would benefit greatly if they could combine their medical data with the insurance records of the patients, but the insurance companies will not be willing to disclose their database, protecting their clients' privacy.

The subject of *privacy-preserving data mining* attempts to address this issue by providing ways of executing the data mining analyses in such cases without the need for either party to completely disclose its database to the other. This paper gives a brief overview of some privacy-preserving data mining methods. The motivation and the statement of the problem are given first. Then the two kinds of techniques—cryptographic and statistical—are briefly presented together with small examples.

2 General Notions

2.1 Motivation

There are two conceptually different contexts of application for privacy-preserving data mining. The first one, that of a *secure multi-party computation* was already mentioned above. More abstractly, suppose there are n *parties*, each party i holding some data D_i. The parties are willing to compute the result of applying some function f to their data: $f(D_1, D_2, \ldots, D_n)$, but none of them is willing to disclose their data to the others. In our case the function f might be some specific data-mining algorithm, such as clustering or association-rule mining.

The second application is related to demography studies. For example, national census surveys collect a lot of confidential information. Once collected, the information must be released to the public in a form that is both privacy-preserving and useful for statistical analyses. Formally, the problem is of transforming a given database D to another form D', which hides private information, but retains some "safe" statistical content. Another example is a web-based survey that collects confidential information from lots of people to estimate certain global statistics. If we modify each surveyed person's data in a consistent manner, we might preserve people's privacy and still retain a way to obtain the required summaries. This exam-

ple is a special case of transforming a database to a "privacy-preserving" form with the assumption that the database is transformed on a record-by-record basis.

2.2 The Two Approaches

There are two approaches to privacy-preserving data mining that more-or-less correspond to the two application areas described above. The secure multi-party computation problem is usually solved using the so-called *cryptographic* approach, the inspiration for which was originally provided in the work [2] by Yao. Effective cryptographical techniques exist that allow to infer decision trees [3], association rules [4] and perform clustering [5] without the need for any party to disclose any information to others. A common trait of these techniques is that they mostly modify the way in which the data-mining algorithms are run but leave the data untouched.

The demographics flavor of the privacy-preserving mining, on the other hand, is better solved by modifying the data itself: by perturbing it randomly, hiding or inserting new values in a certain manner. The technique is usually referred to as *randomization* [6]. In contrast to the cryptographic approach, the randomization methods do not attempt to modify the algorithm. Thus they do not usually require much additional computation and are applicable in an off-line setting. The price for these advantages is somewhat reduced preservation of privacy.

2.3 Distributed vs Centralized Data

The data-mining algorithms considered in this paper process data organized in a tabular manner. We use the term *database* here to refer to such data. A database consists of *rows* (also called *records* or *transactions*) and *columns* (also called *attributes*). Depending on the application, the database might be either *centralized* or *distributed* among several parties. Although we could theoretically imagine the data to be distributed in an arbitrary manner, in practice there are only two common kinds of database partitions. We say that the database is *vertically partitioned* if every party has values for a certain subset of attributes of each record. We call the database *horizontally partitioned* if each party holds a subset of rows of the whole database.

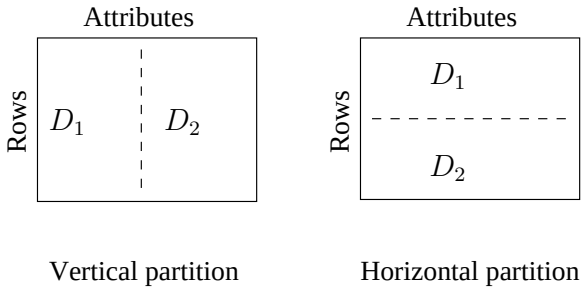

Figure 1: Kinds of database partitions

Vertical partition corresponds to the case of a store that has the billing information of its customers, but lacks data about their income. Horizontal partition corresponds to the case where two different stores have different customers, but hold the same kind of information about them. In both cases the complete database $D = D_1 \cup D_2$ is more useful for knowledge discovery than the parts. This paper will only illustrate the case of horizontally partitioned databases, however, methods exist that handle vertical [4, 5, 7, 8] as well as arbitrary data partition [9].

3 Cryptographic Techniques

Cryptographic techniques approach the multi-party case with distributed data and attempt to derive methods with which the parties may calculate some common result without giving away any of their private information. A simple illustration to this idea is the following *secure-sum* algorithm.

3.1 Secure Sum

Suppose that each of n parties (denote them as $P_1, P_2, \ldots, P_n$) has a secret number u_i, which is a non-negative integer, and the parties would like to calculate the sum of their numbers $u = \sum_i u_i$ without revealing the numbers themselves. In order to achieve it they can act in the following way:

1. Assume $u \in [0, m-1]$. Select P_1 as the *master*.
2. The master generates a random number R uniformly from $[0, m-1]$ and sends the modulo-m sum of this number with its own secret,

$(R + u_1)$, to P_2. It is easy to see that if $R \sim U(0, m-1)$ then $(R + u_1) \sim U(0, m-1)$ (all additions here and further on are modulo-m) therefore P_2 learns nothing about the secret of P_1.

3. P_2 now adds its own value u_2 to the received one and sends it further to P_3. Analogously to the previous case, the value $(R + u_1 + u_2)$, received by P_3 discloses no information. P_3 adds its own secret and passes the value on.

4. When the value reaches the last party P_n it adds its secret u_n and returns the collected sum $R + \sum_{i=1}^{n} u_i$ to the master P_1. The master may now substract R from this value and thus obtain the sum of interest.

3.2 Defining Privacy

The secure sum algorithm presented above, although simple, readily illustrates the major possibilities and problems of the approach. First, note that although no party did explicitly reveal its secret, the obtained sum does leak some information about the possible values for u_i. For example, if there are three parties, the sum u is 60 and $u_1 = 60$, then P_1 will immediately find out that $u_2 = u_3 = 0$. If there are only two parties, then the resulting sum always reveals the secret u_i of the other party. This leak of information is inevitable if the parties wish to learn the output. Therefore we define "privacy" by limiting the information that is leaked by the algorithm to precisely the information that can be learned from the output. This corresponds to the *ideal scenario*, where the parties simply forward their data to a single independent trusted party, that then performs the calculation privately and sends the back the obtained results.

Also note that the sum algorithm preserves privacy assuming certain "honesty" of the participants. If, for example, two out of the three parties agreed to use 0 instead of their secret numbers, they would be able to learn the value of the third participant. There are many other possibilities for an active adversary to disrupt privacy by deviating from the protocol. On the other hand, the algorithm is completely secure against an adversary that correctly follows the protocol, but might try to learn something from the received messages. We call the former kind of adversaries *malicious*, and the latter — *semi-honest*. It is much harder to design an algorithm secure against malicious adversaries, so most methods assume semi-honest participants. This is not too strict a limitation, however, as it often corresponds to the real-life setting. Sometimes it is possible to transform the semi-honest-secure protocol to a malicious-adversary-secure by incorporating zero-knowledge proofs of the legitimacy of each step.

It is rarely the case where absolute security can be achieved without any assumptions about the power of the adversaries. The secure-sum algorithm, for example, will only be secure if the adversary cannot eardrop on the connections between the participants. Many other algorithms require even stronger assumptions, the most common of which is the assumption of a *polynomial adversary*. Stated simply, the assumption requires that the adversary will only be able to run polynomial-time algorithms on its data. This excludes the possibility of simple brute-force attacks.

We shall present here a more elaborate example of a privacy-preserving decision-tree construction algorithm. But before that we shall introduce the notions of oblivious transfer and secure function computation. These will provide some additional insightful illustrations to the whole subject and will be used as primitives in the decision-tree algorithm.

3.3 1-out-of-2 Oblivious Transfer

An important primitive used in many privacy-preserving solutions is the *1-out-of-2 oblivious transfer (OT)* protocol. Suppose that there are two parties: $\mathcal{A}$ and $\mathcal{B}$. Let $\mathcal{B}$ have two pieces of information: y_0 and y_1. $\mathcal{A}$ wishes to learn one of them, y_k. A 1-out-of-2 OT protocol is a protocol involving $\mathcal{A}$ and $\mathcal{B}$ at the end of which $\mathcal{A}$ learns y_k and nothing more, and $\mathcal{B}$ learns nothing (i.e. it does not learn k). Here is a simple example how this protocol could be implemented using the El Gamal public-key encryption scheme:

- Let G be a cyclic multiplicative group satisfying the DDH assumption[1] and let g be any

[1] A cyclic multiplicative group is said to satisfy the *Decisional Diffie-Hellman (DDH) assumption* iff for a randomly chosen generator g and exponents a, b and c the tuples (g, g^a, g^b, g^{ab}) and (g, g^a, g^b, g^c) are *computationally indistiguishable.*

generator of G.

- $\mathcal{B}$ choses a random $c \in G$ and sends it to $\mathcal{A}$.
- $\mathcal{A}$ generates a private key $x \in G$ and a public key $y = g^x$. It also generates another "public key" $z = cy^{-1}$. The computation of the private key corresponding to z should be computationally infeasible due to DDH.
- Suppose $\mathcal{A}$ wishes to perform the oblivious transfer of item $k \in \{1, 2\}$. Let $c_k = y$, the public key for which $\mathcal{A}$ has the corresponding private key, and let $c_{3-k} = z$ the second public key generated by $\mathcal{A}$. $\mathcal{A}$ sends the pair (c_1, c_2) to $\mathcal{B}$.
- $\mathcal{B}$ checks that $c_1 c_2 = c$ (this ensures that $\mathcal{A}$ knows only one of the two private keys), encrypts y_1 using the key c_1, y_2 using key c_2 and sends the two encryptions (e_1, e_2) back to $\mathcal{A}$[2]
- $\mathcal{A}$ decrypts e_k using the available private key x thus obtaining y_k.[3] $\mathcal{A}$ cannot learn y_{3-k} because it cannot decrypt e_{3-k}.

The described protocol is rather effective, requiring only three passes of communication[4], and it is secure if the DDH assumption holds for a chosen $G = \mathbb{Z}_q$.

3.4 Secure Function Computation

Let $f(x, y)$ be a function of two arguments. Let $\mathcal{A}$ be the holder of x, and $\mathcal{B}$ be the holder of y. $\mathcal{A}$ and $\mathcal{B}$ are willing to compute $f(x, y)$ without disclosing their values. This can be done using the protocol proposed by Yao [2].

The main idea is to represent the function f as a combinatorial circuit. The simplest possibility is to express each bit of $f(x, y)$ as a boolean function of the argument bits, and encode the resulting bit-functions in a single boolean circuit.

It is possible to represent any function $f : \{0,1\}^n \to \{0,1\}^m$ in such a way and often the representation is reasonably small (however, there are cases where the size of the resulting circuit is exponential in n).

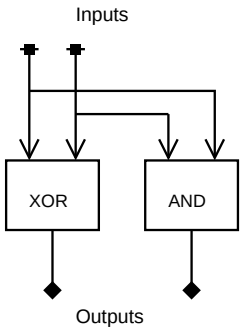

Figure 2: A boolean circuit representation of a function that adds two bits (a half-adder).

The resulting circuit is then *garbled* by $\mathcal{B}$. Garbling is performed in the following way:

- For each wire i in the circuit, $\mathcal{B}$ generates two random numbers: $w_i^{(1)}$ and $w_i^{(0)}$.
- For each gate in the circuit, $\mathcal{B}$ constructs a *table* that maps the garbled input values to the encryptions of the corresponding output values. For example, for the *AND*-gate with input wires i, j and output wire k the following table will be created:

Wire i	Wire j	Encrypted wire k
$w_i^{(0)}$	$w_j^{(0)}$	$E_{w_i^{(0)}, w_j^{(0)}}(w_k^{(0)})$
$w_i^{(0)}$	$w_j^{(1)}$	$E_{w_i^{(0)}, w_j^{(1)}}(w_k^{(0)})$
$w_i^{(1)}$	$w_j^{(0)}$	$E_{w_i^{(1)}, w_j^{(0)}}(w_k^{(0)})$
$w_i^{(1)}$	$w_j^{(1)}$	$E_{w_i^{(1)}, w_j^{(1)}}(w_k^{(1)})$

  where $E_{a,b}(c)$ denotes the encryption of c using keys a and b. This table now allows to determine the garbled value of the output wire of the gate, knowing the garbled values of the input wires: the output value may be obtained by decrypting the corresponding table entry with the input values as the keys.

$\mathcal{B}$ sends to $\mathcal{A}$ the whole garbled circuit (i.e. the tables of all the gates), the garbled values for $\mathcal{B}$-s input wires, and the translation of the garbled values to actual bits for the circuit's output wires.

[2] $e_i = (f_{i1}, f_{i2})$ where $f_{i1} = g^{r_i}$, $f_{i2} = y_i c_i^{r_i}$, and r_i is generated uniformly at random.

[3] $y_k = f_{k2}(f_{k1}^x)^{-1}$

[4] There exist more effective algorithms, however. For example the protocol by Aiello, Ishai and Reingold [10] requires two passes only.

Now if $\mathcal{A}$ could find out the garbled values for its input wires, it could compute the output of the circuit and nothing else. So, in order to obtain the garbled values for each of its input wires, $\mathcal{A}$ invokes a 1-out-of-2 OT protocol with $\mathcal{B}$.

This protocol turns out to be secure in the sense that it does not leak any information apart from that inferrable from the result. It is also reasonably effective: the amount of communication is proportional to the number of gates in the circuit, which is small for most "reasonable" circuits (like addition, multiplication, comparison). It thus allows to securely compute a wide range of functions. However, it is not applicable to typical data-mining problems directly, because although any statistical analysis might be represented as a function f that takes a database of records as its input, the size of the input for this function and its circuit representation are enormous, thus this algorithm is no more practical.

Therefore, more specific methods must be invented for every data-mining algorithm, and such methods do exist. Let us consider the privacy-preserving version of the ID3 decision-tree construction algorithm.

3.5 ID3 in brief

ID3 is a simple and robust method for constructing decision trees, proposed by Quinlan [11]. The derivatives of this algorithm such as C4.5 [12] are widely used in practice.

Let D be some database. For simplicity we assume that the data contains only nominal attributes (i.e. attributes with finite value sets). One of the attributes is special and is representing the *class* of the corresponding record. A *decision tree* is a model of a classifier, that can predict the value of the class attribute, given the values of the other attributes of any record. It is a rooted tree, each non-leaf node of which corresponds to a test on the value of an attribute, and each leaf node specifies the value of the class attribute for all records that would pass the tests on the path from the root to this node. ID3 is an algorithm that constructs decision trees from data. The tree is constructed so as to describe the data in a given database D well. The algorithm proceeds recursevely in a greedy manner, choosing for the nodes of the tree primarily those attributes that best discriminate the class attribute. More specifically:

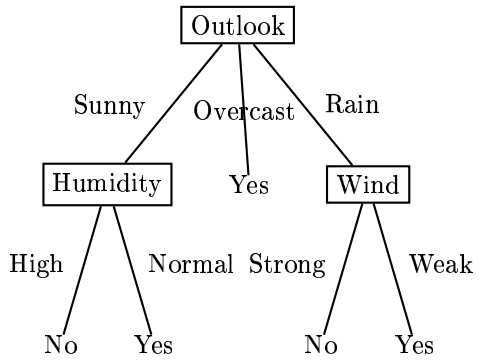

Figure 3: An example of a decision tree that classifies weather conditions into suitable and non-suitable for playing tennis (from [13]).

Algorithm: ID3
Input: a database D. Output: a decision tree.

1. If D is empty, return an empty tree.
2. If all the transactions in D have the same class value, return a leaf node indicating this class value.
3. Otherwise, determine the attribute A that *best* classifies the transactions in D.
4. Create a new tree node N, that splits the data on the values of this attribute.
5. For each value a_i of the attribute A:
   - Select the subset of database records, that have a_i as the value for this attribute, denote the set by $D(a_i)$.
   - Run **ID3**$(D(a_i))$ creating the whole subtree.
6. Return the tree with root node N, edges corresponding to values of attribute A and such that the edge labeled by a_i leads to a subtree returned by **ID3**$(D(a_i))$.

The detail that was left unspecified in the above algorithm description is the method of finding the attribute that best predicts the class value. Such attribute can be found as the attribute having the maximal *mutual information* with the class attribute $I(A;C)$. Mutual information of attribute

A with the class attribute C is often called *information gain* of A and is defined as:

$$\text{Gain}(A) = H_D(C) - H_D(C|A)$$

where $H_D(C)$ is the entropy of the class distribution in the database and $H_D(C|A)$ is the conditional entropy of the class, given A:

$$H_D(C) = -\sum_k P_D(c_k) \log P_D(c_k)$$
$$H_D(C|A) = \sum_i P_D(a_i) H_D(C|A = a_i)$$
$$H_D(C|A = a_i) = -\sum_k P_D(c_k, a_i) \log P_D(c_k, a_i)$$

where $P_D(c_k)$ denotes the fraction of records in D with class c_k, $P_D(c_k, a_i)$ denotes the fraction of records in $D(a_i)$ with class $C = c_k$ and $P_D(a_i)$ is the fraction of records in D with value a_i for attribute A.

Intuitively, the information gain of A shows how many bits of information the value of A discloses on average about the class.

3.6 Privacy-preserving ID3

Suppose the two parties, $\mathcal{A}$ and $\mathcal{B}$ have different records of a horizontally split database D, and are willing to run the ID3 algorithm on D in a privacy-preserving manner. Due to the structure of ID3, it is enough to show how the parties can locate the attribute with the maximum gain at each step, as this is the only action that is performed recursively in the whole algorithm. The result of this intermediate computation may be immediately disclosed to both parties because it is inferrable from the final result of the computation.

3.6.1 Finding the Attribute with Maximum Gain

Let us fix some attribute A. Now note that instead of maximizing gain $H_D(C) - H_D(C|A)$ we might as well minimize the conditional entropy $H_D(C|A)$. Now

$$H_D(C|A) = \sum_i \frac{|D(a_i)|}{|D|} \times$$
$$\times \sum_k -\frac{|D(a_i, c_k)|}{|D(a_i)|} \log \frac{|D(a_i, c_k)|}{|D(a_i)|}$$

where $|\cdot|$ denotes set cardinality and $D(a_i, c_k)$ is the subset of records with $A = a_i$ and class $C = c_k$. So:

$$H_D(C|A) = \frac{1}{|D|} \times$$
$$\times \left(-\sum_i \sum_k |D(a_i, c_k)| \log |D(a_i, c_k)| + \right.$$
$$\left. + \sum_k |D(a_i)| \log |D(a_i)| \right)$$

As we are searching for the attribute that minimizes $H_D(C|A)$, we may ignore the constant $\frac{1}{|D|}$ and use natural logarithms instead of logarithms to base 2. What is left is a sum of the form

$$\sum x \ln x$$

where the values x are the counts of certain types of records in the whole database. Each x is either $|D(a_i)|$ or $|D(a_i, c_k)|$ and is therefore a sum of a value known by $\mathcal{A}$, and a value known by $\mathcal{B}$: for example $|D(a_i)| = |D_1(a_i)| + |D_2(a_i)|$ where $|D_1(a_i)|$ is obtained by counting the number of records with $A = a_i$ in the database held by $\mathcal{A}$ and $|D_2(a_i)|$ is the number of records with $A = a_i$ in the database of $\mathcal{B}$. Therefore if we can manage to privately calculate the sum of the form

$$\sum (x + y) \ln(x + y)$$

we shall also be able to execute the ID3 algorithm privately. But we might use the Yao's universal algorithm to perform this task, as well as the task of comparing such sums obtained for every attribute and selecting the best one. Of course, the circuit for computing $x \ln x$ might be large and the precision rather crude, but this problem is solved using a clever optimization trick that we shall not cover here as it is a bit too involved. The interested reader is referred to the paper [14].

This completes the presentation of the cryptography-based techniques for privacy-preserving data mining. The next part briefly describes the other flavor of the subject: randomization.

4 Randomization

4.1 Numerical Randomization

As already noted before, randomization of data provides alternative ways of preserving privacy. Consider a simple example: people submit their age to a central database via a web-based questionnaire. The collected data is then used to find the average age of the customers. Now note that if we asked each person to add a random value with zero mean to his age prior to submitting it, we might still be able to calculate the average age with acceptable precision while preserving the privacy of the customers. This idea may be generalized and formalized to the following: let each client C_i, $i = 1, 2, \ldots, N$, have a numerical attribute x_i. Assume that all the values x_i are instances of i.i.d. random variables X_i with distribution F_X. The server wishes to learn this distribution, but it must not learn the client's individual values x_i. The clients submit to the server the perturbed values $z_i = x_i + y_i$ where y_i are random shifts with distribution F_Y which is known to the server. The server can also estimate the distribution F_Z of the submitted values z_i. The task is to restore the unknown distribution F_X knowing F_Z and F_Y. The paper [15] by R. Agrawal proposes an iterative algorithm for restoring F_X, based on the Bayes' rule. The algorithm is the following:

1. Let f_X^0 be the uniform distribution. This will be our first approximation to density of F_X.
2. Let j denote the iteration number. At first $j := 0$.
3. Iteratively enhance the approximation:
   - $f_X^{j+1}(a) = \frac{1}{N} \sum_{i=1}^{N} \frac{f_Y(z_i - a) f_X^j(a)}{\int_{-\infty}^{\infty} f_Y(z_i - z) f_X^j(z) dz}$
   - $j := j + 1$

Obviously, the correct f_X is one of the fixpoints of the algorithm. Whether it is the only point is not clear, however. The practical implementation of the algorithm might use piecewise constant functions to represent the densities f_X^j.

This algorithm of density restoration may now be used in a more complex setting. For example, we saw that the decision-tree construction algorithm ID3 only requires the knowledge of the distributions of the attributes. It can be therefore adapted to a randomized setting. Here the server will have to run the "derandomization" algorithm every time it wishes to estimate the distribution of any attribute or class. The article [15] states that this approach is effective for the training sets as large as 100000 records, and provides a decent amount of privacy.

4.2 Privacy Breaches

The randomization-based approach differs from the cryptographical one in the fact that it by definition cannot conceal all the information and something will definitely leak. Therefore some measure of privacy needs to be introduced. The simplest idea is to use the length of the shortest $c\%$ confidence interval $|I(x_i)|$ that the server can establish for the clients' secret values x_i, knowing the submitted values z_i. If the length of the interval is large enough, we might consider the algorithm safe. Unfortunately, this measure can be misleading. One problem is that it is not dependent on the distribution F_X. Consider for example the following density function:

$$f_X(x) = \begin{cases} 0.5 & \text{if } 0 \le x \le 1 \text{ or } 4 \le x \le 5 \\ 0 & \text{otherwise} \end{cases}$$

Assume that the shift Y is distributed uniformly in $[-1, 1]$. The length of the 100% confidence interval is then 2. However, if we take into account that X must always belong to the set $[0, 1] \cup [4, 5]$ we can always compute a shorter confidence interval of length 1. Another example: suppose we are perturbing the age of the people by adding to it a random integer distributed uniformly in $[-50, 50]$. The length of the 100% confidence interval is here 100 and clearly such a randomization does in general a good job hiding information. However, in some particular cases it still allows the server to infer the client's values with reasonable certainty. For example, if the randomized value is 125 then with probability more than 90% the original age is close to 75. Similarly, if the randomized value is -45, the original value must be between 1 and 5. Such situations are called *privacy breaches*, and have to be taken into account when using or designing a randomization-based algorithm.

Another possibility to measure privacy is related to mutual information $I(X; Z)$. The greater this value, the more information about X is retained in

Z and hence the less privacy is preserved. However, even this information-theoretic privacy measure does not free us from the notion of privacy breaches.

5 Summary

To conclude, there exist effective cryptographic and randomization-based solutions that allow to run data-mining algorithms on large datasets and at the same time protect the privacy of the individual data items. The methods are not universal, however, and each new data-mining algorithm will have to be manually converted for the privacy-preserving setting. The existing methods, however, cover most of the typical data-mining tasks and are readily applicable in practice.

The subject is young and there are no well-known textbooks yet. There are some really good overviews, however, such as [1, 14, 4, 6].

References

[1] Vassilios S. Verykios, Elena Bertino, Igor Nai Fovino, Loredana Parasiliti Provenza, Yucel Saygin, and Yannis Theodoridis. State-of-the-art in privacy-preserving data mining, 2003.

[2] A. C. Yao. How to generate and exchange secrets. In *Proc. IEEE Symp. on Foundations of Computer Science*, 1986.

[3] B. Pinkas. Cryptographic techniques for privacy-preserving data mining. 2002.

[4] J. Vaidya and C. Clifton. Privacy preserving association rule mining in vertically partitioned data, 2002.

[5] J. Vaidya and C. Clifton. Privacy-preserving k-means clustering over vertically partitioned data, 2003.

[6] A. Evfimievski. Randomization in privacy-preserving data mining. 2002.

[7] Jaideep S. Vaidya. Privacy preserving data mining over vertically partitioned data.

[8] Boris Rozenberg and Ehud Gudes. Analysis of two approaches for association rules mining in vertically.

[9] Geetha Jagannahan and Rebecca N. Wright. Privacy-preserving distributed k-means clustering over arbitrarily partitioned data, 2005.

[10] Bill Aiello, Yuval Ishai, and Omer Reingold. Priced oblivious transfer: How to sell digital goods. *Lecture Notes in Computer Science*, 2045:119–??, 2001.

[11] J. R. Quinlan. Discovering rules from large collections of examples: a case study.

[12] John Ross Quinlan. C4.5: Programs for machine learning., 1993.

[13] Tom Mitchell. *Machine Learning.* McGraw Hill, 1997. 414 pages.

[14] Yehuda Lindell and Benny Pinkas. Privacy-preserving data mining.

[15] R. Agrawal and R. Srikant. Privacy-preserving data mining, 2000.