Cryptographic techniques for Privacy-Preserving Data Mining

Alexey Vyskubov

23.09.2003

Alice and Bob owns confidential databases.

They want to run a data mining algorithm against the union of databases.

The want to do it in “private” manner.

An ideal scenario: the data mining algorithm is run by trusted third party.

The aim: build a protocol with the same level of "privacy" as in ideal scenario.

The setting: Alice and Bob are semi-honest — honest but curious.

The setting: Authenticity of input data is somehow checked.

1-out-of-2 oblivious transfer.

- Sender: (x_0, x_1)
- Receiver: $\sigma \in \{0, 1\}$
- Receiver learns x_σ
- The idea: receiver uses two public keys; for one the private key is known and for one it the private key is not known.

Oblivious polynomial evaluation involves a sender and a receiver. The sender's input is a polynomial Q of degree k over some field $\mathcal{F}$ and the receiver's input is an element $z \in \mathcal{F}$.

The receiver learns $Q(z)$.

(We will call the sender as Alice and the receiver as Bob.)

The given function should be represented as circuit with gates defined over, e.g., all functions $g : \{0, 1\} \rightarrow \{0, 1\}$.

Any polynomial-time function can be expressed as a circuit of polynomial size.

Alice assigns two random values for every wire in circuit —"garbled" values. They should be long enough to use as a keys to pseudo-random function.

For every gates Alice prepares a table which allows to find garbled output of gate if garbled inputs are known, revealing no other information.

- There is an entry in table for every pair of input values.
- Entries are marked by labels, not by actual (garbled) values.
- The (garbled) value of output wire is encrypted using the (garbled) values of input wires.

Alice sends wiring of the original circuit, gate tables and tables, which allow to convert garbled values of output wires to normal values to Bob.

Bob gets garbled values for his inputs using 1-out-of-2 oblivious transfer.

Bob computes the circuit.

Database is a set of *transactions* and each column is an *attribute* taking on different values.

One of the attributes is designated as a *class* attribute; the set of possible values for this attribute being the classes.

We wish to predict the class of a transaction using only non-class attributes.

The ID3 algorithm solves classification problem by building decision tree — rooted tree containing nodes and edges.

Each internal node is a test node and corresponds to an attribute.

The edges leaving the node correspond to the possible values of the attribute.

The leaves contain the expected class value.

ID3 algorithm is recursive.

At each step the database is partitioned according to the “best” attribute.

How to choose the “best” attribute?

$$H_C(T) = \sum_{i=1}^{l} -\frac{|T(c_i)|}{|T|} \log \frac{|T(c_i)|}{|T|}$$

$$H_C(T \mid A) = \sum_{j=1}^{m} \frac{|T(a_j)|}{|T|} H_C(T(a_j))$$

$$\mathrm{Gain}(A) = H_C(T) - H_C(T \mid A)$$

We can implement ID3 only with some precision δ.

$$|\mathrm{Gain}(A_1) - \mathrm{Gain}(A_2)| < \delta$$

This is not a big problem.

The only difficult step — finding the attribute with maximal gain.

It is enough to minimise $H_C(T \mid A)$.

$$H_C(T \mid A) = \sum_{j=1}^{m} \frac{|T(a_j)|}{|T|} H_C(T(a_j))$$

$$= \frac{1}{|T|} \sum_{j=1}^{m} |T(a_j)| \sum_{i=1}^{l} -\frac{|T(a_j, c_i)|}{|T(a_j)|} \log \frac{|T(a_j, c_i)|}{|T(a_j)|}$$

$$= -\frac{1}{|T|} \sum_{j=1}^{m} \sum_{i=1}^{l} |T(a_j, c_j)| \log |T(a_j, c_j)| +$$

$$\frac{1}{|T|} \sum_{j=1}^{m} |T(a_j)| \log |T(a_j)|.$$

So we need to calculate $(v_1 + v_2) \log(v_1 + v_2)$.

Remark: $1/|T|$ is not important. We also can use ln instead of log.

Alice owns v_1, Bob owns v_2. The goal: Alice obtains w_1 and Bob obtains w_2 s.t.

1. $w_1 + w_2 = (v_1 + v_2) \ln(v_1 + v_2) \bmod |\mathcal{F}|$
2. w_1 and w_2 are uniformly distributed in $\mathcal{F}$ when viewed independently of one another.

Why not Yao's algorithm? It's ineffective, the size of the circuit is of the order of multiplication of the size of its inputs.

Let's start with computing random shares u_1 and u_2, s.t. $u_1 + u_2 = \ln(v_1 + v_2)$.

$$\ln(1+\varepsilon) \approx \sum_{i=1}^{k} \frac{(-1)^{i-1}\varepsilon^i}{i}, -1 < \varepsilon < 1$$

Error shrinks exponentially.

Let 2^n be the power of 2 which is closest to x. Then $x = 2^n(1+\varepsilon)$ and

$$\ln x = n \ln 2 + \varepsilon - \frac{\varepsilon^2}{2} + \frac{\varepsilon^3}{3} - \frac{\varepsilon^4}{4} + \dots$$

Let N be large ($N > \log |T| (> n)$).

Alice constructs small circuit which inputs v_1, v_2 and random α_1 and β_1 and outputs $\alpha_2 = \varepsilon 2^N - \alpha_1$ and $\beta_2 = 2^N n \ln 2 - \beta_1$.

Alice constructs the polynomial (w_1 — random).

$$Q(x) = \operatorname{lcm}(2,\ldots,k)\sum_{i=1}^{k}\frac{(-1)^{i-1}}{2^{(N(i-1))}}\frac{(\alpha_1+x)^i}{i} - w_1,$$

Bob gets $w_2 = Q(\alpha_2)$ using oblivious polynomial evaluation.

$w_1 + w_2$ is an approximation of $\ln\varepsilon$ up to multiplicative factor (which is public and can be removed). Let

$$u_i = w_i + \operatorname{lcm}(2,\ldots,k)\beta_i.$$

Then

$$u_1 + u_2 \approx \operatorname{lcm}(2,\ldots,k)2^N \ln\varepsilon + \operatorname{lcm}(2,\ldots,k)2^N n \ln 2 =$$

$$\operatorname{lcm}(2,\ldots,k)2^N \ln x.$$

So

$$v_1 + v_2 = x, u_1 + u_2 \approx \ln x.$$

Alice defines (r_1, r_2 — random):

$$P_1(y) = v_1 y + r_1, P_2(y) = u_1 y + r_2.$$

Bob runs oblivious polynomial evaluation and obtains

$$P_1(u_2) + P_2(v_2) + u_2 v_2 =$$

$$v_1 u_2 + r_1 + u_1 v_2 + r_2 + u_2 v_2 =$$

$$(v_1 + v_2)(u_1 + u_2) - v_1 u_1 - r_1 - r_2.$$

Alice sets her share to be $v_1 u_1 + r_1 + r_2$.

Now it is not a problem to find the “best” attribute.