A Privacy Preserving Markov Model for Sequence Classification

Suxin Guo  
Department of Computer Science and Engineering  
SUNY at Buffalo  
Buffalo, 14260, U.S.A.  
suxinguo@buffalo.edu

Sheng Zhong  
State Key Laboratory for Novel Software Technology  
Nanjing University  
Nanjing, 210023, China  
sheng.zhong@gmail.com

Aidong Zhang  
Department of Computer Science and Engineering  
SUNY at Buffalo  
Buffalo, 14260, U.S.A.  
azhang@buffalo.edu

ABSTRACT

Sequence classification has attracted much interest in recent years due to its difference from the traditional classification tasks, as well as its wide applications in many fields, such as bioinformatics. As it is not easy to define specific "features" for sequence data as in traditional feature based classifications, many methods have been developed to utilize the particular characteristics of sequences. One common way of classifying sequence data is to use probabilistic generative models, such as the Markov model, to learn the probability distribution of sequences in each class.

One thing that should be considered in the research of sequence classification is the privacy issue. In many cases, especially in the bioinformatics field, the sequence data contains sensitive information which obstructs the mining of data. For example, the DNA and protein sequences of individuals are highly sensitive and should not be released without protection. But in the real world, data is usually distributed among different parties and for the parties, training only with their own data may not give them strong enough models. This raises a problem when some parties, each holding a set of sequences, want to learn the Markov models on the union of their data, but do not want to reveal their data to others due to the privacy concerns. In this paper, we address this problem and propose a method to train the Markov models, from the ones of the first order to the ones of order \( k \) where \( k > 1 \), on sequence data distributed among parties without revealing each party’s private sequences to others. We apply the homomorphic encryption to protect the sensitive information.

Categories and Subject Descriptors

E.3 [Data]: Data Encryption—public key cryptosystems; I.5.2 [Pattern Recognition]: Design Methodology—classifier design and evaluation; J.3 [Computer Applications]: Life and Medical Science—biology and genetics

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

BCB ’13, September 22 - 25, 2013, Washington, DC, USA  
Copyright 2013 ACM 978-1-4503-2434-2/13/09 ...$15.00.

1. INTRODUCTION

Sequence classification has been a hot topic in many fields, including bioinformatics, text mining, speech recognition, and others. In this work, we focus on the classification of symbolic sequences where a sequence is an ordered list of symbols drawn from a finite alphabet, such as DNA and protein sequences. For example, protein sequences are composed of symbols from an alphabet of 20 amino acids. The task of sequence classification is to train a classifier which assigns class labels to sequences.

In traditional feature based classification tasks, a sample typically has a vector of features that represents it. But for the sequence data, the features are not explicit, so that the traditional feature based classifiers cannot be directly used in sequence classification. Many methods have been developed to take advantage of the particular characteristics of sequences to improve the classification performance. One common way of doing this is to train probabilistic generative models on sequence data. Markov model is one of the most popular models because it well captures the probability distribution of each class, and also because of its cost efficiency and decent accuracy.

There is a problem in sequence classification that the privacy issues should be taken into consideration, especially in the bioinformatics field. The DNA and protein sequences are usually collected from particular individuals, and thus contain sensitive information regarding those people, such as genetic markers for diseases [19]. Because of this, the sequences are mostly anonymized. However, even after the anonymization, the sequences still suffer from the threat of re-identification. For example, in many cases, a sequence can be de-anonymized and linked to its human contributor by the recognition of certain markers [19].

The same type of sequence data is usually generated or collected by not only one organization. It is more likely that the data is distributed among different organizations. If the organizations just use their own data to learn the classifiers, there is no privacy violation. But this is not practical because with the limited data, the learned model may not be strong enough. A more reasonable way is that the organi-
zations collaborate with each other and learn the models on the union of their data. Here comes the privacy problem and no one is willing to reveal his/her data to others.

For example, since various kinds of cancer are related with the mutation in the human protein sequences, medical institutes may collect both normal and mutated protein sequences from their patients and volunteer donators, so that they can learn on the data. Then for new coming sequences, the institutes can identify whether they have mutation or not. Obviously, it is better for the institutes to cooperate with each other and learn on the union of their data because they can get stronger models in this way. But the private information within their sequences may stop them from sharing the data.

In this paper, we address the problem of learning Markov models on sequence data distributed among different parties with privacy concerns and propose a method to learn the models without revealing each party’s sequences. We not only deal with ordinary Markov models of the first order, but also extend the method to preserve privacy for the Markov models of order $k$ where $k$ is larger than one. The reason why we extend the method is that [22] has shown that Markov models with higher order can improve the accuracy of sequence classification.

The rest of this paper is organized as follows: We present the related work in Section 2 and the technical preliminaries in Section 3, which includes the background knowledge about the Markov model and the cryptographic tools we need. The details of our method is explained in Section 4. In Section 5, we show the experimental results and finally, Section 6 concludes the paper.

2. RELATED WORK

In recent decades, people have been gradually aware of the privacy problems lay in data analyzing methods. A lot of data mining and machine learning algorithms have been extended to be privacy preserving. Most of these approaches fall into two categories. The methods in the first category protect privacy by data perturbation techniques, such as randomization [1, 17], rotation [4] and resampling [16]. As the original data is perturbed, this kind of methods usually suffer from certain accuracy loss. The approaches of the second category apply cryptographic techniques to protect data during the computations [23, 10]. As the sensitive information is encrypted rather than changed in these approaches, there is typically no accuracy loss. Our work is based on the second way, and applies homomorphic encryptions to protect data.

In the cryptographic category, some secure building blocks are very commonly used, such as secure sum [5], secure comparison [6, 7], secure division [8], secure scalar product [5, 8, 12], secure matrix multiplication [14, 9, 15], etc. The data mining and machine learning algorithms that have been enhanced with privacy solutions include decision tree classification [23, 28], k-means clustering [30, 18], gradient descent methods [31], and others. Actually our work is not the first one that considers the privacy problem about the Markov model. [22] has proposed a method to outsource Markov chains securely without revealing sensitive information. But our problem setting is different from theirs. They consider the scenario that the Markov model has already been learned, and known to one party. Another party has the test queries which are going to be tested against the model. Both of the two parties encrypt their own information and send them to an untrusted server which performs the testing procedure securely. While in our case, the Markov model is not known at the beginning and our goal is to learn the model with training data distributed among different parties. All the computations are done by the data owners, not a server. Hence, the method from [22] cannot be directly applied to our setting.

3. TECHNICAL PRELIMINARIES

3.1 Markov Model For Sequence Classification

We briefly introduce the Markov model and how it is used for sequence classification. We start with the ordinary first order Markov model and then explain the model of order $k$ where $k>1$.

3.1.1 Markov Model of the First Order

We have a set of states of size $m$, which is denoted by $\Sigma$ and we can consider it as an alphabet. A Markov chain is a sequence of states with the Markov property, which means, each state is only dependent on its previous state, not any others, where each state is from the state alphabet. For a sequence $S$ of length $n$, we denote the i-th element in $S$ by $s_i$ and the value of the i-th element by $s_i$ [2]. So that each $s_i$ is from the state alphabet. With the Markov property, we have:

$$P(S_{i+1} = s_{i+1} | S_i = s_i, S_{i-1} = s_{i-1}, \ldots, S_1 = s_1) = P(S_{i+1} = s_{i+1} | S_i = s_i).$$

That is, the probability of state $s_{i+1}$ given all the previous states is the same as the probability of state $s_{i+1}$ given only state $s_i$. Thus, the probability of sequence $S$ is:

$$P(S) = P(S_n = s_n | S_{n-1} = s_{n-1})P(S_{n-1} = s_{n-1} | S_{n-2} = s_{n-2}) \ldots P(S_1 = s_1) = P(S_1 = s_1) \prod_{i=2}^{n} P(S_i = s_i | S_{i-1} = s_{i-1}).$$

We simplify the notation of the above equation as follows:

$$P(S) = P(s_n | s_{n-1})P(s_{n-1} | s_{n-2}) \ldots P(s_1) = P(s_1) \prod_{i=2}^{n} P(s_i | s_{i-1}). \quad (1)$$

To train the Markov models for sequence classification, each element in the alphabet of the sequences is considered as a state. For example, in the classification of protein sequence data, each of the 20 amino acid is treated as a state and $\Sigma$ is the set of amino acids of size 20. We need to calculate these probabilities:

For any state $s_a$ in the state alphabet, its prior probability is:

$$P(s_a) = \frac{\text{count}(s_a) \sum_{s_j \in \Sigma} \text{count}(s_j)}{\sum_{s_j \in \Sigma} \text{count}(s_j)},$$

where $\text{count}(s_a)$ is the number of times $s_a$ appearing in the training set and $\sum_{s_j \in \Sigma} \text{count}(s_j)$ is the sum of the number of times that all the states in the alphabet appear in the
training set, which, in this case, is the total size of the training set, or the sum of lengths of all the sequences in the set.

For any two states $s_a$ and $s_b$ in the state alphabet, the transition probability that $s_b$ happens given $s_a$ is:

$$P(s_b|s_a) = \frac{\text{count}(s_a, s_b)}{\text{count}(s_a)}.$$ 

where count$(s_a, s_b)$ is the number of times $s_a$ is followed by $s_b$ in the training set.

After we calculate the transition probability of every pair of states in the alphabet $\Sigma$, we can get an $m$ by $m$ transition matrix. Then the training process is completed. To test a sequence against a Markov model and examine how likely that this sequence is generated from this model, we just need to follow Equation 1 and calculate the product of all the needed probabilities which can be found in the transition matrix and the priors.

For each class, such a Markov model is trained on the data of this class. Then to test a sequence and identify its class model of order 1 such that each state is dependent on its previous state, or the sum of lengths of all the sequences in the training set.

Then the training process is completed. To test a sequence and identify its class model and the test sequences are tested with every class model.

### 3.1.2 Markov Model of Order $k$

The Markov model of order $k$ is an extension of the Markov model of order 1 such that each state is dependent on its previous $k$ states, not just 1.

With the extension, the probability of sequence $S$ is changed to:

$$P(S) = P(s_1, s_2, \ldots, s_k) \prod_{i=k+1}^{n} P(s_i|s_{i-1}, s_{i-2}, \ldots, s_{i-k}).$$

In this case, the priors are not the probabilities of every single state, but the probabilities of every $k$-gram. Here a $k$-gram means a combination of $k$ symbols from the alphabet. For any $k$-gram $s_1, \ldots, s_k$, its prior probability is:

$$P(s_1, \ldots, s_k) = \frac{\text{count}(s_1, \ldots, s_k)}{\text{count}(\text{all } k\text{-grams})},$$

where count$(s_1, \ldots, s_k)$ is the number of times that $k$-gram $s_1, \ldots, s_k$ appears in the training set at any position and count$(\text{all } k\text{-grams})$ is the sum of the number of times that all possible $k$-grams appear in the training set.

For any state $s_a$ and any $k$-gram $s_1, \ldots, s_k$, the transition probability that $s_b$ happens given $s_1, \ldots, s_k$ is:

$$P(s_b|s_1, \ldots, s_k) = \frac{\text{count}(s_1, \ldots, s_k, s_b)}{\text{count}(s_1, \ldots, s_k)},$$

where count$(s_1, \ldots, s_k, s_b)$ is the number of times $s_1, \ldots, s_k$ is followed by $s_b$ in the training set.

In this case, the transition matrix is not of size $m$ by $m$, but of size $m^k$ by $m$, because the number of all possible $k$-grams is $m^k$. The following procedure is the same as the first order Markov model. We also train a model for each class and the test sequences are tested with every class model.

### 3.2 Privacy Protection of the Markov Model

We assume that each party has a set of sequences and they want to learn the Markov models collaboratively on the union of their data. We develop our secure solution under the semi-honest model, which is widely used in articles of this area [13, 23, 20, 29, 31, 5, 8, 32]. In this model, the parties are assumed to be "honest but curious", which means that the parties follow the protocols correctly, but they would try to derive the private information of others with the intermediate results they get during the execution of protocols. This is a reasonable assumption in the privacy preserving data mining problems, because the goal of all the parties is to get the accurate mining results, so they are not willing to corrupt the protocols and get invalid results.

### 3.3 Cryptographic Tools

#### 3.3.1 Homomorphic Cryptographic Scheme

In this paper, we apply an additive homomorphic asymmetric cryptographic system to perform the encodings and decodings of the data. In an asymmetric cryptographic system, we have a pair of keys: a public key for encryption and a private key for decryption. We denote the encryption of integer $x_1$ by $E(x_1)$. A cryptographic scheme is additive homomorphic if there are operators $\oplus$ and $\odot$ that for any two integers $x_1$, $x_2$ and any constant $a$, we have

$$E(x_1 + x_2) = E(x_1) \oplus E(x_2),$$

$$E(a \times x_1) = a \odot E(x_1).$$

This means, with an additive homomorphic cryptographic system, we can compute the encrypted sum of integers directly from the encryptions of these integers.

#### 3.3.2 ElGamal Cryptographic system

There are several additive homomorphic cryptographic schemes [32, 26]. In this work, we apply a variant of the ElGamal scheme [11], which is semantically secure under the Diffe-Hellman Assumption [3].

ElGamal cryptographic system is a multiplicative homomorphic asymmetric cryptographic system. With this system, the encryption of an integer $f$ is such a pair:

$$E(f) = (f \times g^r, g^r),$$

where $g$ is a generator, $x$ is the private key, $y$ is the public key that $y = g^x$ and $r$ is a random integer.

We call the first part of the pair $c_1$ and the second part $c_2$ so that $c_1 = f \times g^y$ and $c_2 = g^r$. To decrypt $E(f)$, we compute $s = c_2 = g^{ex} = g^{e x_1} = y^r$. Then do $c_1 \times s^{-1} = f \times y^r \times y^{-r}$ and we can get the cleartext $f$.

In the variant of ElGamal scheme we use, the integer $f$ is encrypted in such a way:

$$E(f) = (g^{f^x} \times y^r, g^r).$$

The only difference between the original ElGamal scheme and this variant is that $f$ in the first part is changed to $g^f$. With the change, this variant is an additive homomorphic cryptosystem such that:

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

$$E(a \times x_1) = E(x_1)^a.$$

To decrypt $E(f)$, we follow the same procedure as in the original ElGamal algorithm. But because of the change, after the above decryption, we get $g^{f^x}$ instead of $f$. To obtain $f$ from $g^{f^x}$, we have to perform an exhaustive search, which is to try every possible $f$ and look for the one that matches $g^{f^x}$. 

Please note that the time needed for this exhaustive search is reasonable because we only need to search all possible values of the plaintext, which is not a big range in our case.

We assume that the private key is additively shared by all the parties and no party knows the complete private key. The parties need to coordinate with others to do the decryptions and the ciphertexts can be exposed to every party, because no party can decrypt them without the help of others.

The private key is shared in this way: Suppose there are two parties, parties A and B. A has a part of private key, \( x_A \), and B has the other part, \( x_B \), such that \( x_A + x_B = x \), where \( x \) is the complete private key. In the decryption, we need to compute \( s = c_2^x = c_2^{x_A + x_B} = c_2^{x_A} \times c_2^{x_B} \). Party A calculates \( s_A = c_2^{x_A} \) and party B calculates \( s_B = c_2^{x_B} \) so that \( s = s_A \times s_B \). We need to do \( c_1 = s^{-1} = c_1 \times (s_A \times s_B)^{-1} = c_1 \times s_A^{-1} \times s_B^{-1} \). Party A computes \( c_1 \times s_A^{-1} \) and sends it to party B. Then party B computes \( c_1 \times s_A^{-1} \times s_B^{-1} = c_1 \times s^{-1} = y \) and sends it to A. In this way both parties can get the decrypted result. Here since the party B does its decryption part later, it gets the final result earlier. If it does not send the result to A, the decrypted result can only be known to party B. The order of the parties in the decryptions can be changed, so if we need the result to be known to only one party, the party should do its decryption later.

### 3.3.3 Secure Scalar Product Computation

We apply the secure scalar product computation protocol in [12] to compute the scalar product of two vectors. Given the two \( d \)-dimensional vectors \( x = (x_1, x_2, \ldots, x_d) \) from party A and \( y = (y_1, y_2, \ldots, y_d) \) from party B, the protocol securely computes the scalar product \( p_A + p_B = xy = x_1y_1 + x_2y_2 + \ldots + x_dy_d \) that \( p_A \) is with party A and \( p_B \) is with party B.

### 3.3.4 Secure Logsum Computation

In this work, we also need the secure logsum computation proposed in [27]. The input are two \( d \)-dimensional vectors, \( x = (x_1, x_2, \ldots, x_d) \), which is from party A, and \( y = (y_1, y_2, \ldots, y_d) \), which is from party B, such that \( x + y = \log z = (\log z_1, \log z_2, \ldots, \log z_d) \). The output are two additive shares \( s_A \) held by party A and \( s_B \) held by party B that \( s_A + s_B = \log(\sum_{i=1}^{d} z_i) = \log(\sum_{i=1}^{d} 10^{x_i+y_i}) \).

The basic idea of the secure logsum algorithm is:

First, party A computes vector \( 10^{x_i-q} \) where \( q \) is a random number generated by A and party B computes vector \( 10^y \).

Second, the two parties apply the secure scalar product protocol to calculate the scalar product of the two vectors \( 10^{x_i-q} \) and \( 10^y \). The result \( \phi = \sum_{i=1}^{d} 10^{x_i+y_i-q} \) is only known to party B.

Finally, party B computes \( s_B = \log \phi = \log(\sum_{i=1}^{d} 10^{x_i+y_i}) - q \) and party A has \( s_A = q \) so that \( s_A + s_B = \log(\sum_{i=1}^{d} 10^{x_i+y_i}) = \log(\sum_{i=1}^{d} z_i) \).

### 4. PRIVACY PRESERVING MARKOV MODEL FOR SEQUENCE CLASSIFICATION

In this section, we present how to securely learn the Markov models for sequence classification on data distributed between two parties, A and B. It can clearly be extended to the case when the number of parties is larger than two. For simplicity, we just consider the two-party case here. We start with the first order Markov model and then extend it to the Markov model of order \( k \) where \( k > 1 \).

### 4.1 Markov Model of the First Order

As mentioned in Section 3, the training of the Markov model for each class is to count the occurrences of single states and combinations of states in the class and calculate the prior and transition probabilities.

Let \( C \) be the set of all class labels, which is of size \( l \). Then for each class value \( c_j \in C \), we compute the prior probabilities of states and the transition probabilities.

For any state \( s_a \) in the state alphabet, its prior probability in class \( c_j \) is:

\[
P(s_a | c_j) = \frac{\text{count}(s_a, c_j)}{\text{count}(c_j)},
\]

where \( \text{count}(s_a, c_j) \) is the number of times \( s_a \) appearing in the sequences belonging to class \( c_j \) and \( \text{count}(c_j) \) is the sum of the number of times that all the states in the alphabet appear in the sequences belonging to class \( c_j \), which, in this case, is the sum of lengths of all the sequences belonging to class \( c_j \).

When the data is distributed between parties A and B, we have:

\[
\text{count}(s_a, c_j) = \text{count}_A(s_a, c_j) + \text{count}_B(s_a, c_j),
\]

where \( \text{count}_A(s_a, c_j) \) is the number of times \( s_a \) appearing in the sequences belonging to class \( c_j \) in the data of party A, and \( \text{count}_B(s_a, c_j) \) is the number of times \( s_a \) appearing in the sequences belonging to class \( c_j \) in the data of party B.

To get the total occurrence times of \( s_a \), we need to add up the times it appears in both parties.

Similarly, we have:

\[
\text{count}(c_j) = \text{count}_A(c_j) + \text{count}_B(c_j).
\]

So the prior probability of state \( s_a \) in class \( c_j \) is:

\[
P(s_a | c_j) = \frac{\text{count}_A(s_a, c_j) + \text{count}_B(s_a, c_j)}{\text{count}_A(c_j) + \text{count}_B(c_j)},
\]

where \( \text{count}_A(s_a, c_j) \) and \( \text{count}_A(c_j) \) are held by party A and \( \text{count}_B(s_a, c_j) \) and \( \text{count}_B(c_j) \) are held by party B.

Although the two parties can encrypt their own values and exchange them, it is still hard to calculate \( P(s_a | c_j) \) because an additively homomorphic cryptosystem does not support the secure computation of the division operation between two encrypted integers. So we need to calculate \( \log P(s_a | c_j) \) instead of \( P(s_a | c_j) \), which turns the division into a substraction:

\[
\log P(s_a | c_j) = \log(\text{count}_A(s_a, c_j) + \text{count}_B(s_a, c_j)) - \log(\text{count}_A(c_j) + \text{count}_B(c_j)),
\]

Then the problem becomes how to calculate \( \log(a + b) \) where \( a \) is with party A and \( b \) is with party B securely. Here we need to utilize the secure logsum protocol which takes two \( d \)-dimensional vectors \( x = (x_1, x_2, \ldots, x_d) \) from party A and \( y = (y_1, y_2, \ldots, y_d) \) from party B as input where \( x + y = \log z = (\log z_1, \log z_2, \ldots, \log z_d) \) and outputs two additive shares \( s_A \) held by party A and \( s_B \) held by party B that \( s_A + s_B = \log(\sum_{i=1}^{d} z_i) = \log(\sum_{i=1}^{d} 10^{x_i+y_i}) \).

We feed the secure logsum protocol with such two vectors of \( 2 \)-dimension: \( x = (\log a, 0) \) from party A and \( y = (0, \log b) \) from party B. In this case, \( x + y = \log z = (\log z_1, \log z_2) = \log(a + b) \).
(log a, log b). Then the output of the secure logsum protocol should be $s_A + s_B = \log(z_1 + z_2) = \log(a + b)$.

Following this procedure, $\log(count_A(s_a, c_j) + count_B(s_a, c_j))$ and $\log(count_A(c_j) + count_B(c_j))$ are calculated by the two parties with the secure logsum protocol and shared in this way:

$$\log(count_A(s_a, c_j) + count_B(s_a, c_j)) = s_A^1 + s_B^1$$

$$\log(count_A(c_j) + count_B(c_j)) = s_A^2 + s_B^2,$$

where $s_A^1$ and $s_A^2$ are held by party A and $s_B^1$ and $s_B^2$ are in party B. Then we have:

$$\log P(s_a | s_b) = (s_A^1 + s_B^1) - (s_A^2 + s_B^2)$$

$$= (s_A^1 - s_B^2) + (s_B^1 - s_A^2).$$

$s_A^1 - s_B^2$ can be computed by party A and $s_B^1 - s_A^2$ by party B. The two parties then exchange the two values and both of them can get $\log P(s_a | s_b)$ and calculate $P(s_a | s_b)$.

For any two states $s_a$ and $s_b$ in the state alphabet, the transition probability that $s_b$ happens given $s_a$ in class $c_j$ is:

$$P(s_b | s_a, c_j) = \frac{\text{count}(s_a, s_b, c_j)}{\text{count}(s_a, c_j)},$$

where $\text{count}(s_a, s_b, c_j)$ is the number of times $s_a$ is followed by $s_b$ in the sequences belonging to class $c_j$.

Following the same procedure as the prior probabilities, both parties can get the transition probabilities securely:

$$P(s_b | s_a, c_j) = \frac{\text{count}_A(s_a, s_b, c_j) + \text{count}_B(s_a, s_b, c_j)}{\text{count}_A(s_a, c_j) + \text{count}_B(s_a, c_j)}.$$  

With all the prior probabilities and transition probabilities computed, both parties can get the Markov models of every class. Since the models are known, every party can test its own sequences against the models by itself. The training process of the privacy preserving Markov model of the first order is summarized in Algorithm 1.

### 4.2 Markov Model of Order $k$

The training process of the Markov model of order $k$ follows the same pattern as the training process of the Markov model of order 1.

For any $k$-gram $s_1, \ldots, s_k$, its prior probability in class $c_j$ is:

$$P(s_1, \ldots, s_k | c_j) = \frac{\text{count}(s_1, \ldots, s_k, c_j)}{\text{count}(all \ k - grams \ in \ c_j)},$$

where $\text{count}(s_1, \ldots, s_k, c_j)$ is the number of times that $k$-gram $s_1, \ldots, s_k$ appears in the sequences belonging to class $c_j$ at any position and $\text{count}(all \ k - grams \ in \ c_j)$ is the sum of the number of times that all possible $k$-grams appear in the sequences belonging to class $c_j$.

The two parties can compute the probability securely from their counts with the same method as in the training of the first order Markov model:

$$P(s_1, \ldots, s_k | c_j) = \frac{\text{count}_A(s_1, \ldots, s_k, c_j) + \text{count}_B(s_1, \ldots, s_k, c_j)}{\text{count}_A(all \ k - grams \ in \ c_j) + \text{count}_B(all \ k - grams \ in \ c_j)}.$$  

### Algorithm 1 Privacy Preserving Markov Model of Order 1

**Input:** Party A has a set of sequences $D_A$, and party B has a set of sequences $D_B$.

**Output:** The Markov models of every class, where each model contains the prior probabilities of every state and the transition matrix.

1: **for** each class $c_j$ **do**
2: Party A counts the sum of the number of times that all the states in the alphabet appear in the sequences in $D_A$ that belong to class $c_j$, which is $\text{count}_A(s_a, c_j)$, and party B counts $\text{count}_B(s_a, c_j)$ from $D_B$ in the same way;
3: **for** each state $s_a$ in the state alphabet **do**
4: Party A counts the occurrence times of $s_a$ in the sequences in $D_A$ that belong to class $c_j$, $\text{count}_A(s_a, c_j)$, and party B counts $\text{count}_B(s_a, c_j)$ from $D_B$ in the same way;
5: Parties A and B jointly compute the logarithm of the prior probability of $s_a$ in $c_j$, $\log P(s_a | c_j)$, with the counts they have, under the help of the secure logsum protocol, and then compute $P(s_a | c_j)$;
6: **for** each state $s_b$ in the state alphabet **do**
7: Party A counts the number of times $s_a$ is followed by $s_b$ in the sequences in $D_A$ that belong to class $c_j$, $\text{count}_A(s_a, s_b, c_j)$, and party B counts $\text{count}_B(s_a, s_b, c_j)$ from $D_B$ in the same way;
8: Parties A and B jointly compute the logarithm of the transition probability that $s_b$ happens given $s_a$ in class $c_j$, $\log P(s_b | s_a, c_j)$, with the counts they have, under the help of the secure logsum protocol, and then compute $P(s_b | s_a, c_j)$;
9: **end for**
10: **end for**
11: **end for**

For any state $s_a$ and any $k$-gram $s_1, \ldots, s_k$, the transition probability that $s_a$ happens given $s_1, \ldots, s_k$ in class $c_j$ is:

$$P(s_a | s_1, \ldots, s_k, c_j) = \frac{\text{count}(s_1, \ldots, s_k, s_a, c_j)}{\text{count}(s_1, \ldots, s_k, c_j)},$$

where $\text{count}(s_1, \ldots, s_k, s_a, c_j)$ is the number of times $s_1, \ldots, s_k$ is followed by $s_a$ in the sequences belonging to class $c_j$.

The probability can be computed by:

$$P(s_a | s_1, \ldots, s_k, c_j) = \frac{\text{count}_A(s_1, \ldots, s_k, s_a, c_j) + \text{count}_B(s_1, \ldots, s_k, s_a, c_j)}{\text{count}_A(s_1, \ldots, s_k, c_j) + \text{count}_B(s_1, \ldots, s_k, c_j)}.$$  

The training process of the privacy preserving Markov model of order $k$ is summarized in Algorithm 2.

### 5. EXPERIMENTS

The experimental results are presented in this section. All the algorithms are implemented with the Crypto++ library in the C++ language and the communications between parties are implemented with socket API. The experiments are conducted on a Red Hat server with 16 x 2.27 GHz CPUs and 24G of memory.

We use two real-world datasets to test our algorithms. The first dataset, which is from [25], is a set of inorganic
Algorithm 2 Privacy Preserving Markov Model of Order k

Input: Party A has a set of sequences $D_A$, and party B has a set of sequences $D_B$;

Output: The Markov models of every class, where each model contains the prior probabilities of every $k$-gram and the transition matrix;

1: for each class $c_j$ do
2: Party A counts the sum of the number of times that all possible $k$-grams appear in the sequences in $D_A$ that belong to class $c_j$, which is $count_A(\text{all } k - \text{grams in } c_j)$, and party B counts $count_B(\text{all } k - \text{grams in } c_j)$ from $D_B$ in the same way;
3: for each possible $k$-gram $s_1, \ldots, s_k$ do
4: Party A counts the occurrence times of $s_1, \ldots, s_k$ in the sequences in $D_A$ that belong to class $c_j$, $count_A(s_1, \ldots, s_k, c_j)$, and party B counts $count_B(s_1, \ldots, s_k, c_j)$ from $D_B$ in the same way;
5: Parties A and B jointly compute the logarithm of the prior probability of $s_1, \ldots, s_k$ in $c_j$, $\log P(s_1, \ldots, s_k | c_j)$, with the counts they have, under the help of the secure logsum protocol, and then compute $P(s_1, \ldots, s_k | c_j)$;
6: for each state $s_a$ in the state alphabet do
7: Party A counts the number of times $s_1, \ldots, s_k$ is followed by $s_a$ in the sequences in $D_A$ that belong to class $c_j$, $count_A(s_1, \ldots, s_k, s_a, c_j)$, and party B counts $count_B(s_1, \ldots, s_k, s_a, c_j)$ from $D_B$ in the same way;
8: Parties A and B jointly compute the logarithm of the transition probability that $s_a$ happens given $s_1, \ldots, s_k$ in class $c_j$, $\log P(s_a | s_1, \ldots, s_k, c_j)$, with the counts they have, under the help of the secure logsum protocol, and then compute $P(s_a | s_1, \ldots, s_k, c_j)$;
9: end for
10: end for
11: end for

Table 1: Errors in the Probabilities and the Classification Results

<table>
<thead>
<tr>
<th>Magnitude</th>
<th>Probability Error</th>
<th>Classification Error</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Dataset 1</td>
<td>Dataset 2</td>
</tr>
<tr>
<td>$10^4$</td>
<td>3.6 $\times$ 10$^{-4}$</td>
<td>9.1 $\times$ 10$^{-4}$</td>
</tr>
<tr>
<td>$10^5$</td>
<td>2.1 $\times$ 10$^{-4}$</td>
<td>1.5 $\times$ 10$^{-4}$</td>
</tr>
<tr>
<td>$10^6$</td>
<td>2.8 $\times$ 10$^{-7}$</td>
<td>2.0 $\times$ 10$^{-7}$</td>
</tr>
<tr>
<td>$10^7$</td>
<td>4.5 $\times$ 10$^{-6}$</td>
<td>1.8 $\times$ 10$^{-6}$</td>
</tr>
</tbody>
</table>

Table 1 shows that when the magnitude becomes larger, the probability errors become smaller. For people who want to learn perfect models, it seems that a very large magnitude would be a good choice. But there is a problem that large magnitude also causes high computation cost and long training time, so we need to find a balance between the accuracy and efficiency. Table 2 presents how the running time increases with the magnitude. We get these time durations by training a first order Markov model for one class on each of the two datasets.

The running time is affected not only by the magnitude, but also by the order of the Markov models. When the value of $k$ increases, the training time of a Markov model of order $k$ also increases. Table 3 shows the training time of a Markov...
model of order $k$ for the cases that $k = 1$ and $k = 2$. For each $k$, the total training time of a model is $t$; the time of doing the secure logsum calculations is denoted by $t_l$ and the time of other communications is denoted by $t_c$. All the times in Table 3 are obtained when the magnitude is set to $10^3$.

Table 2: Running Time Affected by the Magnitude

<table>
<thead>
<tr>
<th>Magnitude</th>
<th>Running Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Dataset 1</td>
</tr>
<tr>
<td>$10^0$</td>
<td>70s</td>
</tr>
<tr>
<td>$10^1$</td>
<td>70s</td>
</tr>
<tr>
<td>$10^2$</td>
<td>76s</td>
</tr>
<tr>
<td>$10^3$</td>
<td>137s</td>
</tr>
</tbody>
</table>

Table 3: Running Time Affected by the Order

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Order 1</th>
<th>Order 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$t_l$</td>
<td>$t_c$</td>
</tr>
<tr>
<td></td>
<td>$t_l$</td>
<td>$t_c$</td>
</tr>
<tr>
<td>Dataset 1</td>
<td>42s</td>
<td>34s</td>
</tr>
<tr>
<td>Dataset 2</td>
<td>50s</td>
<td>34s</td>
</tr>
</tbody>
</table>

We can find from Table 3 that when training a Markov model, the time of doing the secure logsum calculations and other communications $t_l + t_c$ plays a dominant role in the total time $t$. The time of the local operations, such as the parties computing the counts in their own domain, is trivial. Hence, the training time is not very relative to the size of the training data, but is more relative to the value of $k$ and the size of the alphabet, $m$. This is because the number of secure logsum calculations and communications, which dominates the overall time, is determined by the number of probabilities to be computed, including the prior probabilities and the transition probabilities. It can be seen in Section 4 that the calculation of each probability requires one secure logsum computation, and some communications between parties.

The number of probabilities to be computed is determined by $k$ and $m$. The number of prior probabilities for a Markov model of order $k$ is $m^k$, because we need to compute a prior probability for each $k$-gram, and the number of all possible $k$-grams is $m^k$. The number of transition probabilities is $m^k \times m$, because there is a transition probability for every $k$-gram and state pair.

When the value of $k$ increases, the number of probabilities increases exponentially. Denote the number of probabilities when the value of $k$ is $i$ to be $np_i$, then $np_{i+1} = np_i \times m$. Thus, the training time when $k = i + 1$ should also be $m$ times of the training time when $k = i$. Table 3 supports this conclusion that all the times $t_l$, $t_c$, and $t$ in the case of order 2 are around 20 times of their counterparts in the case of order 1, where 20 is the size of the amino acid alphabet shared by the two datasets.

On the other hand, when the value of $k$ is fixed, the difference between the time costed to train a model in the two datasets is not significant, though the sizes of the two datasets are very different. This is because that the size of the data does not affect the running time as much as the size of the alphabet does.

With the above discussions, we can see that it is not affordable for ordinary computers to train a Markov model of order $k$ when $k$ is very large. This problem lies not only in the privacy preserving solution, but also in the original Markov model of order $k$ [2]. Fortunately, the Markov model of order $k$ can give decent accuracy even when $k$ is very small.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed a method that enables two parties to securely train Markov models of order $k$ on the union of their sequence data without revealing either party’s information to the other. We evaluated the method with two real-world datasets and shown that the information loss in our privacy preserving algorithm is very low. We also analyzed the running time of the algorithm. Although we focus on the sequence classification task here, the proposed privacy preserving Markov model method can be extended and used in other fields, and this will be our future work.

7. ACKNOWLEDGMENTS

The materials published in this paper are partially supported by the National Science Foundation under Grants No. 1218393, No. 1016929, and No. 0101244.

8. REFERENCES


