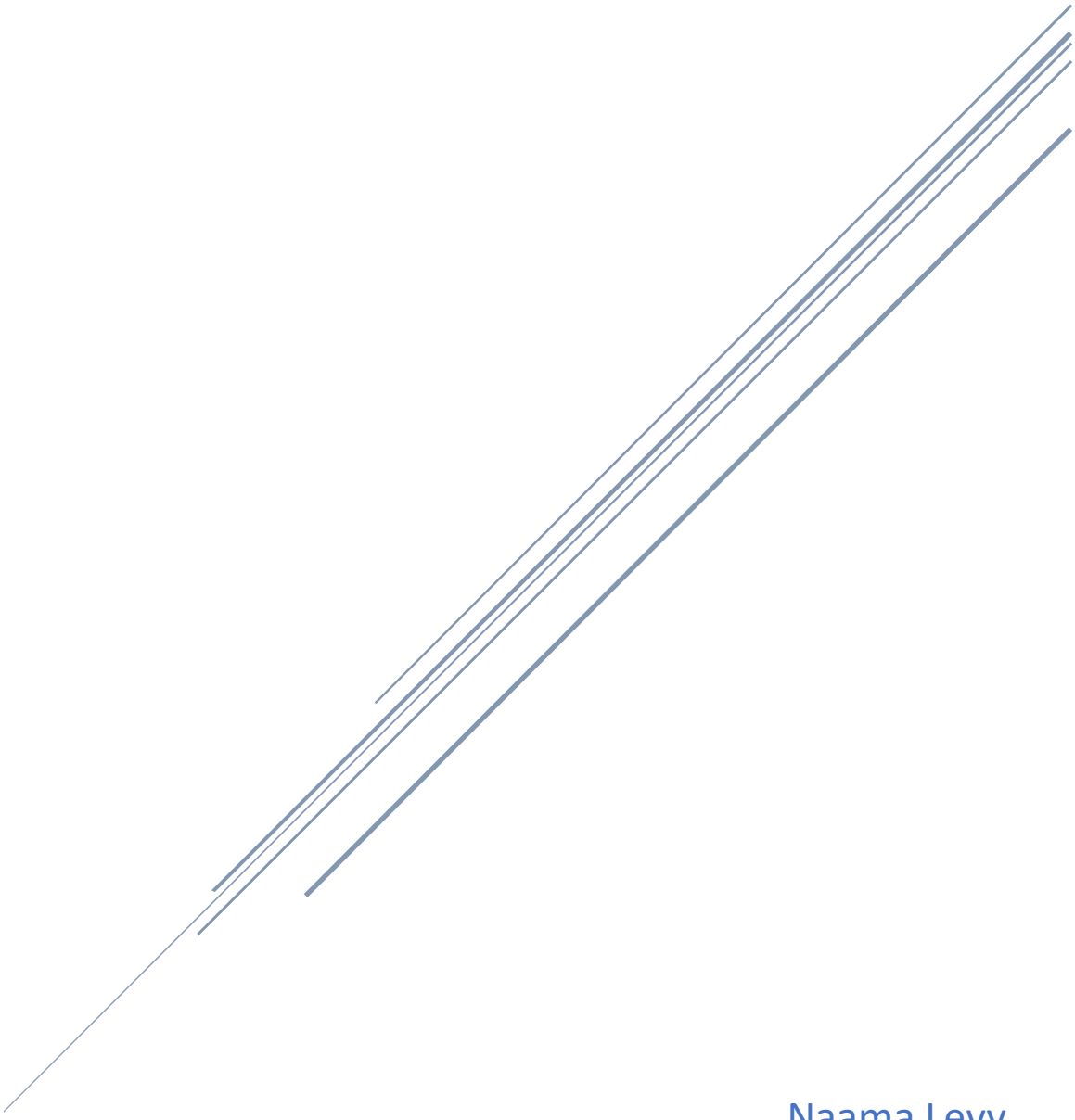
MINI-PROJECT IN PRIVACY PRESERVING DATA MINING

Privacy Preserving Decision Trees



Naama Levy
Yael Geisler

Problem Description

Data mining over multiple data sources has emerged as an important practical problem with applications in different areas, such as data streams, data-warehouses, and bioinformatics. Although the data sources are willing to run data mining algorithms in these cases, they do not want to reveal any extra information about their data to other sources due to legal or competition concerns.

One approach to deal with this problem is using cryptographic methods. However, the computation and communication complexity of such solutions make the use of these methods impractical when many data sources are involved.

Project Introduction

In this project, we wish to build and perform a **solution for data mining over multiple data sources, where each data source does not want to reveal its specific data to other sources**.

Our work is based on the article *'Privacy preserving decision tree learning over multiple parties'*[1].

We consider a scenario where multiple data sources are willing to run data mining algorithms over the union of their data but still preserve their privacy by not letting any of their data be revealed.

We focus on the classification problem in particular, and implement a privacy-preserving algorithm for building a decision tree over an arbitrary number of distributed sources, using the ID3 algorithm.

Algorithm Description

We implemented the ID3 algorithm for building a decision tree. In order to build the tree over multiple parties in a privacy-preserving manner, we used the Shamir Secret Sharing scheme.

Shamir Secret Sharing Scheme

In our implementation of the scheme, we consider a server and a set of n parties $P_1, \dots, P_n$, where each party P_i has a secret value v_i.

The secret sharing algorithm[2] works as follows:

1. The server selects a random set X of n publicly known values $x_1, \dots, x_n$.
2. We define $k = n - 1$ as the degree of the random polynomial.
3. Each party P_i executes the following:
    a. Selects a random polynomial $q_i(x)$ of degree k, such that $v_i = q_i(0)$
    b. For each $1 \leq j \neq i \leq n$, calculates $q_i(x_j)$ and sends to P_j.
    c. For each $1 \leq j \neq i \leq n$, receive $q_j(x_i)$ from P_j.
    d. Computes $S(x_i) = \sum_{j=1}^{n} q_j(x_i)$
    e. Sends $S(x_i)$ to the server.
4. The server receives $S(x_i)$ from each party P_i.
5. $S(x)$ is considered as $\sum_{j=1}^{n} q_j(x)$, thus the server now has n pairs: $[x_i, S(x_i)]$.

6. The server performs Lagrange interpolation to identify coefficients of $S(x)$.
7. The server calculates $S(0) = q_1(0) + \cdots + q_n(0) = v_1 + \cdots + v_n$, which is the sum of all secret values.

Algorithm 2. Privacy preserving summation of secrets, executed at each party P_i

Require: P: Set of parties $P_1, \ldots, P_n$,
v_i: Secret value of P_i,
X: A set of n publicly known random values $x_1, \ldots x_n$ such that $x_i \neq 0$,
k: Degree of the random polynomial
1: Select a random polynomial $q_i(x) = a_{k-1}x^{k-1} + \ldots + a_1x^1 + v_i$
2: /* Distribution phase */
3: Compute the share of each party, P_j, where $sh(v_i,P_j) = q_i(x_j)$
4: **for** $j = 1$ to n **do**
5: Send $sh(v_i,P_j)$ to peer P_j
6: **end for**
7: /* Intermediate computation phase */
8: Receive the shares $sh(v_j,P_i)$ from every party P_j
9: Compute $INTERRES_i = \sum_{j=0}^{n} sh(v_j, P_i)$
10: **for** $j = 1$ to n **do**
11: Send $INTERRES_i$ to peer P_j
12: **end for**
13: /* Final computation phase */
14: Receive the intermediate results $INTERRES_j$ from every party P_j
15: Solve the set of equations to find the $SUM = \sum_{i=0}^{n} v_j$

ID3 Algorithm

We will perform the **Distributed ID3 algorithm** to construct the tree.

ID3 algorithm[3] is a greedy heuristic algorithm that at each iteration selects an attribute A that best classifies the examples (based on a chosen gain function) and divides the examples according to it (by creating a node for attribute A). At the end of its run, we get a decision tree based on the input data.

When using the distributed ID3 algorithm, to keep the summation secret, each party will perform **Shamir's secret sharing algorithm**, compute its classification information for every attribute, contributing its share to each attribute gain function sum as described below:

Gain Function

At each iteration of the algorithm, we choose an attribute A to split the tree according to. We choose the attribute whose gain is maximal.

Let C be the set of all possible categories $c_1, \dots c_k$.

For each category c_i we define:

$T(c_i)$ = The set of objects whose category is c_i

Let A be an attribute and $a_1, \dots, a_p$ its possible values.

For each attribute A, we define:

$T(a_i)$ = The set of objects whose A attribute value is a_i

$T(a_i, c_j)$ = The set of objects whose A attribute value is a_i and category is c_j

Let T be a set of objects whose categories are known. Then the information needed to classify an object in T is:

$$E(T) = \sum_{i=1}^{k} \left(-\frac{T(c_i)}{|T|} \cdot \log \frac{T(c_i)}{|T|} \right)$$

Let us assume that the objects have n attributes $A_1, \dots, A_n$. In order to assess the prediction quality of an attribute A, we need to calculate the information needed to classify an object in T given its value for attribute A. Then the information of T given A is:

$$E(T|A) = \sum_{i=1}^{p} \frac{|T(a_i)|}{|T|} \cdot E\big(T(a_i)\big)$$

Then, the information gain for each attribute A is:

$\mathrm{Gain}(A) = E(T) - E(T|A)$.

The best attribute A is the one that has the maximal gain; thus, it is the attribute A with the **minimal** $E(T|A)$ among all considered attributes.

When dealing with distributed data (of several parties), we need to be able to determine $E(T|A)$ for each attribute among all data sources. we use the equations and their proves described in the paper and calculate $E(T|A)$ as follows:

$$E(T|A) = \sum_{i=1}^{p} \frac{|\ T(a_i)\ |}{|\ T\ |} \cdot E(T(a_i)) = \frac{1}{|\ T\ |} \sum_{i=1}^{p} \left[|\ T(a_i)\ | \cdot \sum_{j=1}^{k} \left(-\frac{|\ T(a_i, c_j)\ |}{|\ T(a_i)\ |} \cdot \log \left(\frac{|\ T(a_i, c_j)\ |}{|\ T(a_i)\ |} \right) \right) \right]$$

$$= \frac{1}{|\ T\ |} \left[-\sum_{i=1}^{p} \sum_{j=1}^{k} \left(|\ T(a_i, c_j)\ | \cdot \log(|\ T(a_i, c_j)\ |) \right) + \sum_{i=1}^{p} \left(|\ T(a_i)\ | \cdot \log(|\ T(a_i)\ |) \right) \right]$$

Given there are s data sources, $|T(a_i)|$, $|T(a_i, c_j)|$ are calculated as follows:

$$|\ T(a_i)\ | = \sum_{l=1}^{s} |\ T_l(a_i)\ | \quad \text{and} \quad |\ T(a_i, c_j)\ | = \sum_{l=1}^{s} |\ T_l(a_i, c_j)\ |$$

Thus, in each step of the algorithm, we can find the best attribute by summing $T(a_i), T(a_i, c_j)$ values (secret values) from all players for every attribute, every possible value of that attribute, and every category.

We chose to emit the $\frac{1}{|T|}$ as it would be constant for every step of the algorithm run.

The secret values sum is revealed using the Shamir Secret Sum algorithm described above.

Privacy Preserving ID3

Require:

R: the set of attributes to be considered

O: the set of objects to be considered

$C = \{c_1, \dots, c_k\}$: the set of possible categories

ID3 (R, O, C):

1. If all objects in O have the same category c_i:
    a. Return a leaf node whose category is c_i
2. If R is empty:
    a. Return a leaf node whose category is set to the dominant category among the objects in O.
3. Determine the attribute A that best classifies the objects in O and assign it as the test attribute for the current tree node.
4. For each possible value a_i of A:
    a. Return $\mathrm{ID3}(\mathrm{R}\backslash\{\mathrm{A}\}, \mathrm{O} \cap \mathrm{T}(\mathrm{a_I}), \mathrm{C})$

Each calculation requires a sum of values found amongst the different data sources, which is done by the Shamir Secure Sum algorithm.

Software

We implemented the described algorithm in Python. Our program consists of two main classes:

Dealer

Runs the ID3 algorithm for building the decision tree. It is aware of the attributes set (and their possible values) and the categories that each object can be classified to.

Used Libraries:

- random
- log2 from Math library
- lagrange from scipy.interpolate library
- copy

Class Fields:

- __num_of_PLAYERS – the number of different parties.
- __tree – a Node class object, represents the root of the tree to be built.

Main functions:

calc_E_TA(A, Ta, Tac):

- A – a set of possible values a_i for attribute A
- Ta – a list of Ta values for each a_i
- Tac – a list of list holding values of Ta_ic_j for each a_i and each c_i (category)

Calculates and returns the E(T|A) value as described above.

get_sec_val_sum(X, PX):

- X – a list of random values drawn by the dealer. Each value corresponds with one player.
- PX – a list of the polynomial (of the summed polynomials) values for each $x_i \in X$. PX[i] = P(X[i])

The method uses Lagrange interpolation to find the polynomial's coefficients, then sets x to zero and reveals the constant term which is the parties' secret values sum.

get_Tai(attrs):

- attrs – a dict where the keys are attributes and the values are the matching desired values.

The dealer creates a new set of random X, and asks each player to return the number of objects that match the desired attributes. The players return the values hidden, according to Shamir's scheme). It then calls the *get_sec_val_sum* with the X set and its corresponding values for the sum polynomial and returns the result.

get_Tac(attrs, ci)

This method is similar to get_Tai. The difference here is that it asks the user for the number of objects that match the desired attributes and hold category ci.

__ID3(R, C, node):

- R – a set of attributes to be considered
- C – set of possible categories
- node – in the first call, it is set to be the tree root.

A recursive private method that called by the public method ID3. It greedily builds the decision tree, as described above. It has 2 stopping criteria:

1. If all objects in the distributed DB's have the same category, the node value would be set to the category, and this node will be a leaf.
2. If there are no more attributes to consider (for data division), the node value would be set to the max_category, defined to be the most common category in the data.

Else: the algorithm finds the best attribute (with min(E(T|A)) to divide the data according to the chosen attribute's possible values.

- The algorithm iterates all attributes in R and calculates their E(T|A) using the described functions, finds the best one, and sets it to the node value.
- It then iterates through the attributes possible values (children), for each value:
  - creates a node
  - Sets child's attributes (Node's field) to be the node's (parent) attributes and adds the current best attribute value.
  - If there are no data objects with these attributes values, this child node will be a leaf (as described in the first stopping criteria)
  - Else: it updates R (removes the current best attribute), calls ID3 recursively with the child node, and at last, appends the child in the current node children array.

__predict(node, attrs, path=None):

- node – initialized to the tree's root
- attrs – a dict of attributes' key-value, which represent a data object (entry).
- path – a list to hold the visited nodes.

A recursive private method that called by the public method 'predict'.
This method predicts a single entry path in the decision tree, from its root to a leaf representing a classification (category), which was set by the algorithm learning set.
The returned value would be the leaf value – the predicted category for this entry.

Player

Each player simulates a different data source. All players are operated by the dealer.

Used Libraries:

- random

Class Main Fields:

- __db – a private field of type 'DB'. It holds the processed data of the player's data source.
- __poly – a list of the hidden polynomial coefficients.

Main functions:

get_poly_val(x, secret):

- secret – the secret value to be encrypted.
- x – a rando value drawn by the dealer to be placed in the polynomial.

The player uses the secret as the constant term of the polynomial. It then places x in the polynomial and calculates the result.

get_Tai(attrs_dict, x, stop=False):

- attrs_dict – a dict where the keys are attributes, and the values are the matching desired values.
- x – the random known value assigned for this player.
- stop – a boolean defaulting to false on the first iteration. If set to false, it implies the player should gather Tai values from the other players.

The player counts the number of objects that match the desired attributes and uses it as the constant term of the polynomial. It encrypts this value by calling get_poly_val.
If *stop* is set to false, it calls for all other players' *get_Tai(attrs_dict, x, stop=True)*.
It calculates the sum of the values returned by all players and returns it to the dealer.

get_Tac(attrs_dict, c, x, stop=False):

This method is similar to get_Tai. The difference here is that it that the player calculates the number of objects that match the desired attributes and that hold category ci.

Dataset and Preprocessing

The chosen data set is *'Dataset for estimation of obesity levels based on eating habits and physical condition in individuals from Colombia, Peru and Mexico'*[4].
This database consists of 2111 records (entries), each entry represents one individual's numeral values for 17 different attributes and is labeled with one of 7 obesity levels.

General Preprocessing

- First, some of the object attributes were removed (8 left) to make it more lightweight and simple to process.
- We reduced the obesity levels to 4 :
    - Insufficient Weight
    - Normal Weight
    - Overweight Level I, Overweight Level II
    - Obesity Type I, Obesity Type II, and Obesity Type III.
- Then, all continuous values were replaced with discrete values (predefined ranges).
- The modified dataset was split into five equal parts. Four of them would simulate the distributed data source, and the fifth would act as a test sample.

Processes DB's Attributes and Values

Attributes

- Gender: Male, Female
- Age: <20, 21-25, 26-35, 36+
- Weight: <=50, 51-65, 66-80, 81-95, 96+
- Height: <=1.6, 1.61-1.7, 1.71-1.8, 1.81+
- OFH (Overweight Family History): yes, no
- FAVC (Frequent Consumption of High Caloric Food): yes, no
- CAEC (Consumption of food between meals): 0, 1, 2, 3
- FAF (Physical Activity Frequency): 0, 1, 2, 3

Categories

- Insfficient
- Normal
- Overweight
- Obesity

Program-Level Pre-processing

Each player had a private 'db' member.
The db was built as nested dict. The dict keys were all the possible attributes combinations.

The value was a dict itself, where the keys were all possible values combination and the values were their number of occorunces in the database and their partition to categories.

```
{ <attributes-combination>:
                    { <values-combination>:   [ #occorunces,
                                                 { <category>: #category-occorrunces}
                                                ]
                    }
}
```

For example:

User Interuction and Test Results

The program is started by running "main.py".
The program is generalized and can be run with any dataset, by setting the relevant attributes and categories before sending them over to the dealer.
Once the program is started, the tree is constructed and the test sample error is calculated.

Then, the user can start interacting with the console: the user may insert its data and predict its obesity level. In each step, the user is requested to insert the suitable value from the options of the current attribute presented on the screen. In the background, the algorithm ‘travels’ in his path from the tree root to the fitting leaf for the user predicted category.

User Interaction Printouts

```
Tree constructed!

Calculating test db error..
The test db error is 10.21%

Hello, what would you like to do?
Press 1 if you'd like to predict your obesity level, or 2 if you want to leave
1
Enter the corresponding number for the following questions:

What is your gender?
1. Male
2. Female
1
How old are you?
1. <=20
2. 21-25
3. 26-35
4. 36+
1
What is your weight?
1. <=50
2. 51-65
3. 66-80
4. 81-95
5. 96+
1
What is your height?
1. <=1.6
2. 1.61-1.7
3. 1.71-1.8
4. 1.81+
1
Do you have family history of overweight?
1. Yes
2. No
1
Do you eat high caloric food frequently?
1. Yes
2. No
1
Do you eat any food between meals?
1. No
2. Sometimes
3. Frequently
4. Always
1
How often do you have physical activity
1. Never
2. One or two days
3. Two or four days
4. Four or five days
1

Your predicted obesity level is Normal
If want to see your path through the tree, press 1. Else, press 2.
1
node: Weight
    node: Height
        node: CAEC
            node: FAVC
                label: Normal
```

Test Results

At the end of the algorithm run, we have a constructed decision tree. We use the fifth part of the split dataset to test our tree's accurate rate.

The error rate was 10.21% for the test dataset, which means we got 89.79% accuracy.

Evaluation

Assuming we have a total of n entries in the data, classified into c categories. Each entry has l attributes, where each attribute has at most m possible values.

Space Complexity

DB:

Each player holds a DB of the following size: the number of attributes combinations, multiply the possible values for each combination. $\sum_{k=0}^{l}\binom{l}{k}\cdot m^k = \sum_{k=0}^{l}\binom{l}{k}\cdot(1)^{l-k}\cdot m^k = (1+m)^l$ (according to Newton Binomial[5][5])

We have a constant number of players; therefore, all players consume space of $O((1+m)^l)$.

Decision Tree:

The server holds the building tree.

The maximal height for the tree (excluding the leaf nodes) is l. Each node (excluding the nodes at level l) has at most m children. The nodes at level l have leaf-children only (hence, categories), therefore the nodes at level l have at most c children,

Thus, the size of the tree is bounded to: $c\cdot m^l + \sum_{k=1}^{l-1} m^k \leq c\cdot m^l + (1+m)^{l-1} = O(c\cdot m^l)$

Hence, **the total space complexity is** $\boldsymbol{O\left((1+m)^l\right) + O\left(c\cdot m^l\right) \approx O\left(c(1+m)^l\right)}$.

Time Complexity

DB:

Constructing the DB requires iterating all entries in the data. With the use of a dictionary, inserting and fetching values from the DB cost amortized $O(1)$. Thus, the construction of the DB takes $O(n)$.

Decision Tree:

As described above, the size of the tree (excluding the leaf nodes) is bounded by $(1+m)^{l-1}$.

At each iteration we choose the best attribute to split according to, using the secret summation of all the players' secret values, which is done for all possible attributes and their values. This value is bounded by $l\cdot m\cdot c$ for each node.

We then perform Lagrange algorithm to discover the secret sum. Since we have a constant number of players, the degree of the polynomial is constant and the time complexity is $O(1)$ at each iteration.

Since the players hold a DB in the form of a dictionary, each player can fetch their secret value at amortized $O(1)$.

Therefore, the tree construction takes $O\left((1+m)^{l-1}\right)\cdot O(l\cdot m\cdot c) \approx O\left(c\cdot l\cdot m^l\right)$

Hence, **the total space complexity is** $\boldsymbol{O(n) + O\left(c\cdot l\cdot m^l\right) \approx O\left(n + c\cdot l\cdot m^l\right)}$**.**

References

[1] F. Emekci, O. D. Sahin, D. Agrawal, and A. El Abbadi. 2007. Privacy preserving decision tree learning over multiple parties. Data Knowl. Eng. 63, 2 (November, 2007), 348–361.
DOI: https://doi.org/10.1016/j.datak.2007.02.004

[2] Adi Shamir. 1979. How to share a secret. Commun. ACM 22, 11 (Nov. 1979), 612–613.
DOI: https://doi.org/10.1145/359168.359176

[3] J. R. Quinlan. 1986. Induction of Decision Trees. Mach. Learn. 1, 1 (March 1986), 81–106.
DOI: https://doi.org/10.1023/A:1022643204877

[4] Palechor, F. M., & de la Hoz Manotas, A. (2019). Dataset for estimation of obesity levels based on eating habits and physical condition in individuals from Colombia, Peru and Mexico. Data in Brief, 104344

[5] https://encyclopediaofmath.org/wiki/Newton_binomial