UT DALLAS
ANONYMIZATION TOOLBOX

In an effort to promote our research in the area of privacy preserving data analysis, as UT Dallas Data Security and Privacy Lab, we complied our implementation of various anonymization methods into a toolbox for public use by researchers. This document explains basic information on how to invoke these anonymization methods (section 1), various features of our implementation of the methods (section 2 and detailed information for those that would like to extend the toolbox (section 3).

An important design consideration during development of the toolbox has been possible memory issues due to large input sizes. In order to facilitate anonymization of arbitrarily large datasets, the toolbox works through an embedded database (SQLite [1]) to minimize memory requirements.

1 Basics

This section is primarily intended for those that want to anonymize some dataset through our toolbox. We explain the input file format in section 1.1. Section 1.2 provides the details on various output file format choices made available to the user. Then, in section 1.3, structure of the configuration file that specifies identifier, quasi-identifier and sensitive attributes along with generalization hierarchies is discussed. Finally, section 1.4 discusses possible user options in executing the toolbox through command line.

1.1 Input format

Currently, the toolbox only supports unstructured text files. In the near future, we plan to add support for database connectivity, XML files and possibly the RDF data model.

By unstructured text files, we refer to ASCII files where each line corresponds to a record and attribute values of a record are separated by some character sequence (e.g., comma, tab, semi-column), specified in the configuration file or passed as an argument to the toolbox. This structure conforms with the default file format of many data repository (e.g., [2]).

All descriptive information regarding attribute types and indices will be collected through the configuration file. Therefore, unlike [3], a header with such description will not be necessary.

1.2 Output format

By default, output format is the same as the input format. The only difference caused by anonymization is over quasi-identifier attributes through replacement of specific values with their generalizations. We refer to this output format as $genVals$ in the toolbox.

The toolbox also allows outputting anonymized records with additional information, as described in some studies. Among these, the first approach proposed in [4] that tries to increase the accuracy of classification models built on anonymized data by releasing equivalence-wide statistics on each quasi-identifier attribute, called $QI - statistics$. For numerical attributes, such statistics contain the mean and variance across the equivalence class. For categorical attributes, on the other hand, the complete distribution is provided. We refer to this format as $genValsDist$ in the toolbox, with reference to the distribution information padded to quasi-identifier attributes.

The last output format supported by the toolbox approach is "anatomization" described in [5]. Anatomization involves releasing two datasets at the end of anonymization. Both datasets contain as many tuples as there are in the input file. The first table, ST, is built by replacing the quasi-identifier by a new attribute that represents the group (i.e., equivalence class) that the tuple belongs in. The second table QIT contains the same new attribute plus the ground level quasi-identifier values. The idea here is that, the two tables can be joined through the group identifier but the join result cannot be utilized the attack the underlying privacy definition. We refer to this format as *anatomy* in the toolbox.

1.3 Configuration file

The configuration file is structured in XML format[1]. Every configuration starts with a root node $config$. The first field of $config$, if specified, provides the anonymization method choice. Depending on the privacy definition, $config$ might also contain a variety of fields corresponding to privacy parameters. These include k for k-anonymity, l for entropy l-diversity, c for (c, l) recursive l-diversity and t for t-closeness. Notice that setting none, multiple or all of these parameters is allowed, in which case the toolbox would select the appropriate parameter(s) depending on the anonymization method. Alternatively, these parameters can be specified through command line arguments (see section 1.4.

The root node $config$ has at most 5 children specified in arbitrary order:

1. *input*: Input parameters. The $filename$ field is mandatory. Separator field, *separator*, can be skipped. The default value is comma.

2. *output* (optional): Output parameters. The $filename$ field is mandatory. Output format field, $format$, assumes a value from the set $\{genVals, genValsDist, anatomy\}$. If no value is specified, $genVals$ is set by default.

3. *id* (optional): List of identifier attributes. For each attribute, *index* field of the corresponding node, which represents the index of the attribute in the dataset has to be

[1]To distinguish between dataset attributes and XML node's attributes, we refer to the later as a field.

set. These attributes will be removed from the output to ensure anonymity.

4. *qid*: List of quasi-identifier attributes. The field *index* of each node *att* should be set. Also for categorical attributes, a mapping (*map*) from categories to integer values is required. We recommend that the mapping start with 0 and is incremented by 1 for each successive category. The next node, *vgh* that follows is optional[2] and its structure is further discussed below.

5. *sens* (optional): List of sensitive attributes. Again, *index* field of the attribute should be set. For categorical attributes, the user also needs to specify a mapping. Similar to *qid* attribute mappings, it is recommended that the mapping start with the value 0 and is incremented by 1.

The *vgh* nodes of quasi-identifier attributes help specify the value generalization hierarchies (VGHs). Since configuration file is in XML format, the natural way to represent the VGHs is to use a tree structure. Starting with the suppression value, denoted by the root node *vgh* of the VGH, every set of child nodes at the same level corresponds to a generalized domain.

Associated with each node *node* is the field *value* that specifies the range of values that can be generalized to it. Ranges for generalized values are expressed mathematically. For example, $(x : y)$ contains all values between x and y, excluding x and y. Similarly, $(x : y]$ implies the upper bound is inclusive.

An important issue in declaring the VGHs is the question of inclusion of the ground domain (i.e., the original, non-generalized domain). For continuous attributes, there is simply infinitely many values in the ground domain and including all is not possible. Therefore, the VGH should exclude the ground domain. Since we wanted to treat categorical and discrete attributes in a similar fashion, and reduce the tedious work of declaring VGHs, we suggest the same for all attributes. Consequently, most-specific values (from the ground domain) never appear in the VGHs.

Notice that the mapping of categorical quasi-identifier attribute domains from categories to integer values should be performed cautiously so that the generalized domains are consistent with the integer values. We recommend that the integer values be assigned in order of breadth-first traversal of the VGH, starting with 0.

An example configuration over the "census-income" dataset of [2] is provided below. In this configuration, the user specifies only one privacy parameter, $k = 5$ as a field of the root node. Apparently, the privacy definition at hand is k-anonymity[3]. In the below configuration, the 2^{nd} and 3^{rd} attributes are considered identifiers. The 13^{th} and 1^{st} attributes together form the quasi-identifier and the only sensitive attribute is the 42^{th} attribute. The corresponding VGHs are depicted in figure 1.

[2]Not every anonymization method requires a value generalization hierarchy (e.g., Mondrian [6]).

[3]Alternatively, all parameters could be specified, in which case the first line becomes "<config k = '5' l = '3' c = '0.2' t = '0.2' >".

```
<?xml version="1.0"?>
<config method = 'Datafly' k = '5'>
   <input filename='census-income_ALL.data' separator=','/>
   <output filename='census-incomeK5.data' format ='genValsDist'/>
   <id>
      <att index='1' name='att1'/>
      <att index='2' name='att2'/>
   </id>
   <qid>
      <att index='12' name='sex'>
         <map>
            <entry cat='Female' int='0' />
            <entry cat='Male' int='1' />
         </map>
         <vgh value='[1:2]'>
         </vgh>
      </att>
      <att index='0' name ='age'>
         <vgh value='[0:100)'>
            <node value='[0:50)'>
               <node value='[0:25)'/>
               <node value='[25:50)'/>
            </node>
            <node value='[50:100)'>
               <node value='[50:75)'/>
               <node value='[75:100)'/>
            </node>
         </vgh>
      </att>
   </qid>
   <sens>
      <att index='41' name='salary'>
         <map>
            <entry cat='- 50000.' int='0' />
            <entry cat='50000+.' int='1' />
         </map>
      </att>
   </sens>
</config>
```

Input validation is performed while the configuration file is parsed. For example, privacy parameters k, l, t, and c should be assigned positive values. The range for t is $(0, 1)$.

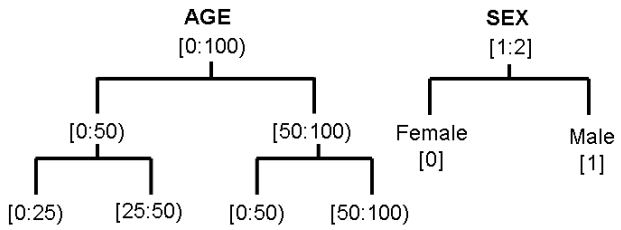

Figure 1: Graphical representation of VGHs of Age and Sex attributes

Parameter	Domain or explanation	Default
-method	Choose from {Datafly, Incognito_k, Incognito_l, Incognito_t, Mondrian, Anatomy}	Mondrian
-config	path to the configuration file	config.xml
-k	any positive integer value	10
-l	any positive real value	10
-c	any positive real value	0.2
-t	any value within the range (0, 1)	0.2
-suppThreshold	max. number of tuples to be suppressed (applies only to Datafly and Incognito_K)	k
-input	input filename	N/A
-separator	character sequence separating attribute values	comma
-output	output filename	N/A
-outputFormat	Choose from {genVals, genValsDist, anatomy}	genVals

Table 1: Parameters that can be set through program arguments

Similarly, for domain-generalization methods, we make sure that every leaf of the VGH is at the same depth.

1.4 Parameters

The configuration file helps specify almost every parameter of anonymization but does not cover all of them. Specifically, unless the configuration file is named "config.xml" (the default value), configuration filename should be passed as a program argument. Additionally, it is possible to set the anonymization method, privacy parameters and input/output parameters from command line as well. Such flexibility facilitates easy batch processing for experimentation purposes.

Table 1 provides the exhaustive list of command line parameters. The first column lists the parameter keys and the second column lists the values that will be recognized by the toolbox. The default value of each parameter is given in the third column.

When executing the toolbox, these parameter keys should be followed by the corresponding values separated by whitespace characters. For example, the argument list

```
-method Datafly -config config.xml -outputFormat anatomy
```

sets the method as Datafly, configuration file as config.xml and output format as *anatomy*. Of course, the order of parameter keys is insignificant as long as the coupling with corresponding values is preserved.

2 Anonymization Methods

The toolbox currently contains 6 different anonymization methods over 3 different privacy definitions. We discuss implementation details of each next.

2.1 Datafly

Datafly was the first algorithm to satisfy k-anonymity privacy definition [7]. The algorithm consists of full-domain generalization until every combination of quasi-identifier values appears at least k times. As described in [7], apart from generalization, another tool is suppression that removes certain tuples altogether from the anonymized dataset[4].

By default, our implementation sets the maximum number of tuples that can be suppressed to k as suggested in [7]. However, through program arguments, this parameter can be reset to another values (maybe 0, if suppression is not desirable).

Since Datafly performs full-domain generalization, upon initialization, the toolbox first validate the VGHs of all quasi-identifier attributes. The check fails if there are any leaf nodes of VGH that are not at the same depth as the others.

2.2 Mondrian Multidimensional k-Anonymity

Full-domain generalization methods (e.g., Datafly) tend to over-generalize the data because generalization is performed at the domain level. Mondrian multidimensional anonymization method solves this problem by performing generalization at the equivalence level [6]. The resulting dataset typically contains more equivalence classes and release more accurate information on quasi-identifier attributes without violating the privacy definition, k-anonymity.

Mondrian operates through an attribute selection heuristic that determines which quasi-identifier attribute an equivalence will be partitioned on. Our implementation in the toolbox uses the heuristic described in section 4 of [6] that "chooses the dimension with the widest (normalized) range of values". When multiple dimensions have the same width, we simply select the first dimension that has an allowable cut.

Once a dimension is chosen, our implementation performs partitioning independently of the corresponding VGH (if specified[5]). Partitioning is performed over the median value such that any values less than or equal to the median resides in the left equivalence and all other in the right equivalence.

[4]Rather than removing the tuples, we set the quasi-identifier attribute values to the lowest granularity value in corresponding VGHs. This approach maintains input/output tuple synchronization.

[5]Effectively, this implies that all VGH specifications are ignored by Mondrian.

2.3 Incognito

Given a set of quasi-identifiers and their VGHs, often there are multiple anonymizations of a dataset that satisfies the privacy definition and respects the domain-generalization requirement. The method Incognito [8] tries to choose the most specific (i.e., least generalized) in an efficient manner.

Various implementation choices are discussed in [8]. Among these, our implementation uses the bottom-up pre-computation optimization described in Section 3.3.2 of the paper.

After the entire generalization lattice is traversed (in breadth-first order), among all successful anonymizations, the toolbox chooses the one that yields the maximum number of equivalence classes. Definitely, this selection heuristic agrees with the approach of choosing the minimum height anonymization described in [8] since generalization can only reduce the number of equivalences.

Similar to Datafly, Incognito can perform suppression as well. Therefore, by default, our implementation sets the suppression threshold parameter of anonymization to k. As discussed before in section 2.1 this functionality can be turned off by setting *suppThreshold* parameter to 0 using program arguments.

2.4 Incognito with l-diversity

k-anonymity privacy definition is vulnerable to adversaries that have strong background knowledge on individuals represented in the dataset. l-diversity tries to overcome such vulnerabilities by diversifying sensitive values within each equivalence class [9].

The toolbox implements two instantiations of l-diversity discussed in [9]. These are entropy l-diversity and recursive (c, l)-diversity. The algorithm determines on a particular instantiation based on whether parameter c is set as a parameter or not. This basically implies that if $c \leq 0$, entropy l-diversity will be assumed and otherwise, recursive (c, l)-diversity.

Our approach to multiple-sensitive attributes relies on the solution described in [9]. All sensitive attributes should be merged into a single attribute, which then is specified as the only sensitive attribute.

Incognito anonymization with k-anonymity as the privacy definition allowed suppression by default, where suppression threshold was set to k. Notice that a similar approach does not apply to l-diversity, since the purpose of anonymization is to diversify the sensitive value distribution. Therefore, suppression is disable within this version of Incognito.

2.5 Incognito with t-closeness

In [10], it is argued that l-diversity is too strong and unnecessary a requirement for datasets with skewed sensitive attribute distributions. Instead, another privacy definition, t-closeness is proposed. t-closeness restrict the distance between the distributions of the sensitive attribute value within the original table and each equivalence. Any table where such distance is above t for at least one equivalence violates the privacy definition.

Our implementation handles only one sensitive attribute, which could be numerical or categorical, but not hierarchical. Data type of the sensitive attribute is inferred from the configuration file (i.e., if a categorical to numeric domain mapping has been specified, then the sensitive attribute is considered categorical).

Similar to our l-diversity implementation with Incognito, suppression option is disabled for t-closeness. Also, as was the case with l-diversity, only one sensitive attribute should be specified.

2.6 Anatomy

Anatomization [5] is another method of generating l-diverse equivalences. In this approach, the privacy definition is a distinct instantiation of the l-diversity concept discussed in [9]. The Anatomization algorithm outputs equivalences of size l such that each equivalence contains at least l well-represented sensitive values.

One major disadvantage of Anatomization is the eligibility requirement [5], which requires that "every sensitive value be associated with at most n/l records", where n is the dataset size. Whenever this condition is not met, our implementation simply outputs an error message and exits.

Another important property Anatomization is that, the output format can only be *anatomy*. The reason behind is quite simple, without generalization in place, quasi-identifier values cannot be represented properly with other output format. As a result, regardless of the parameters specified as program arguments or the configuration file, the toolbox sets the output format as *anatomy*.

3 Extending The Toolbox

We expect the toolbox to be updated frequently, and extended by other researchers in the area. Therefore, in this section, we explain the basic approach that should be taken to writing a new anonymization method. To this end, we discuss the various data structures and objects within the *anonymizer* package. Every anonymization method implemented by the toolbox extends the abstract *Anonymizer* class within this package by overriding certain member functions.

3.1 Importing the source code

As explained before, we work with an embedded version of the SQLite database. Apart from extracting the jar that contains the source code, some work has to be done to adjust system parameters to introduce the correct driver and library.

Fortunately, most necessary commands are provided in the shell scripts included in the toolbox package. More specifically, one needs to provide the SQLite library using the command

```
-Djava.library.path=path-to-sqlite-library-directory
```

where *path-to-sqlite-library-directory* represents the path to directory that contains the SQLite native library (either *sqlite.dll* or *libsqlite.so* depending on the environment).

Also required is the addition of the JDBC driver for the corresponding OS. For Windows systems, include *sqliteWin.jar* and for Linux systems, include *sqliteLinux.jar* to your classpath.

3.2 Data representation

At any time during anonymization, two tables of type $AnonRecordTable$ and $EquivalenceTable$ contain the most current version of generalized data. Among these, the $EquivalenceTable$ has the following schema:

```
EquivalenceTable = {EquivalenceID (auto-increment, key), Interval_1, ...,
    Interval_q}
```

Basically, for each equivalence class, we store an identifier assigned automatically by the database and as many intervals as the number of quasi-identifier attributes. In the above schema, q quasi-identifier attributes is assumed and $Interval\_i$ corresponds to the generalization of quasi-identifier attribute i for that equivalence.

The other table, $AnonRecordTable$, contains all necessary attribute values in ground form, represented by double-precision values. The schema is as follows:

```
AnonRecordTable = {RecordID (auto-increment, key), EquivalenceID,
    QuasiIdentifierAttribute_1, ..., QuasiIdentifierAttribute_q,
    SensitiveAttribute_1, ..., SensitiveAttribute_s}
```

Here, the field $RecordID$ is generated automatically by the database. $EquivalenceID$ attribute references an entry in the $EquivalenceTable$ as a pointer to the current generalization of a tuple. Also included as attributes are quasi-identifier attribute values and sensitive attribute values. If any algorithm does not require or specify sensitive attribute (e.g., Datafly), the last set of attributes can be skipped quite easily.

Although the toolbox removes the database files at the end of execution, it is possible to drop any of these tables during execution. In fact, we recommend dropping all unnecessary tables to reduce the burden on the database.

3.3 Reading the input

Given the input filename, the toolbox reads the records line by line, skipping all lines that contain an unknown value, i.e., an attribute value set to '?'. With every tuple, typically, two insertions are performed:

1. Into a table of type $EquivalenceTable$: First we map categorical attribute values to numerical values using the mappings obtained from the configuration file. Then, we construct intervals based on raw attribute values (e.g., value 1 mapped to interval [1]). Next, we build an equivalence class from these intervals and check whether a similar

entry exists in the table. If it does, no insertion is actually performed and the identifier of the matching equivalence is returned. Otherwise, a new equivalence is inserted and its identifier is returned.

2. Into a table of type *AnonRecordTable*: This step is rather straightforward. The only necessary information is the equivalence identifier obtained from the insertion into *EquivalenceTable* and raw attribute values in numerical form.

3.4 Anonymization

Since all data records are stored in an embedded database, by generating multiple instances of *EquivalenceTable* and *AnonRecordTable* objects, one can actually store multiple anonymizations of the same dataset simultaneously. With many methods, this has been our approach.

Anonymization starts with the initial anonymized record and equivalence tables that result from reading the input into the database. Then, depending on whether the method at hand is iterative (e.g., Datafly) or recursive (e.g., Mondrian), at each step either new database tables are created or existing tables are altered into a new state of generalization.

The only requirements of correctness are that, at the end of the anonymization step:

- *Anonymizer.anonTable* contains all records with the original record identifiers.
- *Anonymizer.eqTable* contains a set of equivalence classes that satisfies the privacy definition.
- Every equivalence identifiers of *Anonymizer.anonTable* references an entry of *Anonymizer.eqTable*.

The outputting of anonymized records is performed separately of anonymization. Notice that while reading input data into the database, we got rid of all irrelevant attributes to improve efficiency. The cost of this approach is, while outputting the results, we need to go back to the input file and fetch the discard attributes. Therefore, in this step, we go through the records in the input file and for each line fetch relevant data from the database.

If the output format is set as $genValsDist$, outputting of anonymized records is preceded by a statistical information collection step, where distribution data of each equivalence are computed.

References

[1] "Sqlite home page," Online, http://www.sqlite.org/.

[2] D. Newman, S. Hettich, C. Blake, and C. Merz, "UCI repository of machine learning databases," 1998.

[3] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: An update," *SIGKDD Explorations*, vol. 11:1, 2009.

[4] A. Inan, M. Kantarcioglu, and E. Bertino, "Using anonymized data for classification," in *ICDE*, 2009, pp. 429–440.

[5] Xiao and Y. Tao, "Anatomy: simple and effective privacy preservation," in *VLDB '06: Proceedings of the 32nd international conference on Very large data bases*. VLDB Endowment, 2006, pp. 139–150.

[6] K. Lefevre, D. J. DeWitt, and R. Ramakrishnan, "Mondrian multidimensional k-anonymity," in *ICDE '06: Proceedings of the 22nd International Conference on Data Engineering (ICDE'06)*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 25–36.

[7] L. Sweeney, "Achieving k-anonymity privacy protection using generalization and suppression," *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, vol. 10, no. 5, pp. 571–588, 2002.

[8] K. LeFevre, D. J. DeWitt, and R. Ramakrishnan, "Incognito: efficient full-domain k-anonymity," in *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*. New York, NY, USA: ACM, 2005, pp. 49–60.

[9] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam, "l-diversity: Privacy beyond k-anonymity," *ICDE 2006*, p. 24, 2006.

[10] N. Li, T. Li, and S. Venkatasubramanian, "t-closeness: Privacy beyond k-anonymity and l-diversity," in *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, 15-20 April 2007, pp. 106–115.