2017 IEEE 6th International Congress on Big Data

Scalable, Efficient Anonymization with INCOGNITO - Framework & Algorithm

Antorweep Chakravorty*, Chunming Rong*, K. R. Jayaram† and Shu Tao†
*Department of Computer Science
University of Stavanger, 4036 Stavanger, Norway
Email: {antorweep.chakravorty,chunming.rong}@uis.no
†Enterprise Cloud Engineering
10598 New York, USA
Email: {jayaramkr,shutao}@us.ibm.com

***Abstract*—With the advent of ”big-data” processing and analytics, organizations and enterprises have increased the collection of data from individuals, and are increasingly developing business models involving analytics to gain deep insights into the collected data. Often, it becomes essential to release and merge said data to third-parties for more extensive analytics for which an organization may not have the necessary expertise. Data often has to be anonymized prior to such release, to safeguard the privacy of individuals involved. While several algorithms, with varying privacy guarantees, have been proposed for anonymizing data, large scale distributed anonymization remains an under-explored topic.**

In this paper, we propose Incognito, a distributed algorithm and framework for anonymization of large data sets. Incognito as a framework is targeted at data center environments, both private data centers and public clouds; and is intended to be compatible with modern data analytics frameworks like map-reduce and resilient distributed datasets (RDDs). Incognito the algorithm aims at minimizing identity, similarity and skins based attacks on anonymized data sets. This paper describes Incognito in detail along with an empirical evaluation of its scalability and efficiency.

***keywords* —anonymization; data privacy; big data; cloud computing**

I. Introduction

Data about individuals have been widely collected over the last decades in different sectors by different institutes, organizations, government agencies and care providers. It is often necessary to release micro data between organizations as well as to the public to serve benign purposes. However, release of such data could often lead to discloser of sensitive personal information to malicious actors. Often, a prescribed strategy has been to remove identifiable information (e.g.. name, social security number) from the released data sets. Still, combinations of selected characteristics of an individual in a population could often allow identification of individuals as demonstrated by L. Sweeney [1]. In the United States, 87% of the population had characteristics that likely made them unique based only on {5-digit zip code, gender, date of birth} in their censors data.

A widely accepted model to protect such characteristics in data has been that of K-Anonymization [1], [2], [3], [4]. It groups k records as subsets such that the values for all identifiable attributes in a subset are the same. In doing so, k-anonymization guarantees against identity discloser [3]. However, it ignores sensitive attributes that specify the outcome/observation about a particular record. Since, the main objective of any anonymization is to protect sensitive information about individuals, k-anonymization becomes vulnerable to attacks arising from an adversary's unavoidable knowledge of the overall distribution of sensitive attribute values in a released table. k-anonymized data sets are liable to skewness and similarity based attacks [5], [6], [7], [8] when their sensitive attribute distribution in a subset is significantly different than that in the original table.

We propose an anonymization algorithm called Incognito, that aims at minimizing identity, similarity and skewness based attacks on anonymized data. It limits the difference in distribution of sensitive attribute values in a released subset to that in the original table by adhering to the taxonomy tree of the sensitive attributes. It also ensures, that the probability change of any sensitive value appearing in a subset remained bounded to an upper bound. A bucketization phase creates overlapping buckets and maintains an upper bound limiting the change in probability of a sensitive attribute value in a released subset generated from these buckets. The sizes for the subsets are dynamically determined by dichotomizing or recursively dividing the buckets sizes into two equal parts. A redistribution phase is also introduced, that generates subsets with sensitive attribute probabilities close to the original data set. Finally, a recording function anonymizes the data using some summary statistical functions such as mean, median, ranges and others to ensure records appearing in each released subsets are identical to each other.

However, with the ever increasing scale of data sets and adaptation of cloud based solutions, the need for anonymizing in some cloud applications increases tremendously [9], [10]. Anonymizing such data sets is becoming a considerable challenge for traditional anonymization algorithms and newer research has started to focused on the scalability problem of large-scale data anonymization [11], [12]. The models proposed for k-anonymization have deteriorating performance with larger data sets [13], [14]. Using distributed processing frameworks performance of such solutions can be vastly improved. Through our Incognito framework, we demonstrate the scalability of our algorithm and two other popular anonymization techniques: t-closeness [5] and β-likeness [8].

In rest of the paper, section II introduces the definitions and background information used through out this paper. Section III, describes the Incognito algorithm. Section IV presents empirical evaluations on the scalability of the framework for different anonymization algorithms (including ours) on large data sets. It also evaluates the identifiability and utility of anonymized data sets using such algorithms. Section V gives the related work and the conclusion is in section VI.

II. Background

The following subsections describe the background information about different terminologies and methods used throughout this paper.

A. Quasi Identifier (QI)

Attributes of type numeric $\{N_1, N_2, \ldots, N_n\}$ and string $\{C_1, C_2, \ldots, C_c\}$ in a private table T that in conjunction with themselves or in relationship with external and publicly available information can re-identify individual records.

B. Sensitive Attribute (SA)

Attributes of type numeric $\{SA_{N1}, SA_{N2}, \ldots, SA_{Nn}\}$ and string
$\{SA_{C1}, SA_{C2}, \ldots, SA_{Cc}\}$ in a private table T that describe results, observations or classifications of records in T. Values for these attributes for any individual should be protected from people who have no direct access to the original data.

C. Equivalence Class (EC)

Let a table $T = \{N_{rn}, C_{rc}\}$; be a set of QI columns. N_{rn} and C_{rc}, respectively being a matrix of type numeric and string attributes. An equivalence class for T with respect to its attributes $\{N_1, N_2, \ldots, N_n\}$ and $\{C_1, C_2, \ldots, C_c\}$ is the set of all records in T containing identical values for all numeric $(x_1, x_2, \ldots, x_n)$ for $N_1, N_2, \ldots, N_n$ and string $(y_1, y_2, \ldots, y_c)$ for $C_1, C_2, \ldots, C_c$ values.

D. Bucketization

Given a table T with SA values, where $SA \subset T$, a set of buckets $\phi = \{B_1, B_2, \ldots, B_{|\phi|}\}$ defined over the SA values such that all SA values having the same value or values having the same parent in their taxonomy tree appear in at least one bucket and $\cup_{i=1}^{n} B_i = SA$.

E. Dichotomization

Again with a table T and a set of buckets ϕ over its SA values, the number and sizes of ECs can be determined by dividing the buckets into equal halves until a proportionality requirement is fulfilled. The proportionality requirement is defined by assuming that an EC, EC_e, is formed with x_i records from bucket $B_i \in \phi, i = 1, 2, \ldots, |\phi|$. EC_e abides to the proportionality requirement with respect to ϕ, if and only if the sizes of x_i are proportional to those of B_i , i.e., $x_1 : x_2 : \ldots : x_{|\phi|} = |B_1| : |B_2| : \ldots : |B_{|\phi|}|$.

F. Upper Bound

The upper bound defines the allowed percentage change of the probabilities of SA values appearing in an EC to that in original table. Given table T with sensitive attribute SA, let $P = (p_1, p_2, \ldots, p_m)$ be the overall SA distribution in T. An EC, EC_e with SA distribution $Q = (q_1, q_2, \ldots, q_m)$ is said to maintain its upper bound, if and only if $\forall q_i, D(p_i, q_i) = q_i - p_i \leq min(\beta, -ln(p_i))$, where β is an user defined allowed probability threshold and $ln(p_i)$ is the natural logarithm of p_i.

G. k-Anonymization

Records in a table T are partitioned into subsets of ECs having at least k records each. All records within an EC are assigned the same, generalized value [15] over each of their QI attributes so that QI values of records in an EC are identical to each other.

III. Incognito Algorithm

The premise of the algorithm is that SA values have a taxonomy tree. Bucketization on sub-trees of SA value taxonomy would allow for creation of ECs that remain close to the original data set. A straightforward approach would be to create buckets for each unique SA value. Having T as the original data set and $\phi \subset T$ containing all the buckets $(B_i; i = 1, 2, \ldots, |\phi|)$ of SA values, where each B_i have the same SA values, then $\cup_{B_i \in \phi} B_i = T$. However, this would lead to a smaller number of ECs with larger sizes. In order to increase data utility, generated ECs need to contain a minimum number of records from an optimal number of buckets to attain balance between data utility and sensitivity. In the following subsections, we describe each of the phases and the respective algorithms.

A. Overlapping Bucketization

We propose a bottom up overlapping bucketization of SA values based on their taxonomy tree, as illustrated in example 1. Given a data set T, the frequency histogram of SA values is generated as a count of SA values at a given level in its taxonomy tree. The probabilities of all SA values are measured for the frequency histogram. For each probability, its upper bound [8] is calculated as $U(p) = 1 + min(\beta, p \times (1 - ln(p)))$; here β is the user specified maximum allowed change threshold (in percent) of SA values appearing in ECs, p the probability of the least frequent value and $ln(p)$ the natural logarithm of the probability. On the level of the leaf nodes, the upper bound is calculated for each SA value. However, as we move to the parent levels, the upper bound is representative of the least frequent SA value for each subset at that level.

Given a set of SA values and their frequency histogram, overlapping buckets can be created for subsets of SA values having the same parent in the taxonomy tree. We start with the level above the leaf nodes and each subset of values in that level is sorted in the ascending order of their frequencies. Next, moving left to right, values in a subset are sequentially iterated. They are grouped as a bucket, provided the sum of probabilities of the values in the bucket is less than or equal to the upper bound. The upper bound is determined as allowed probability change for the value with the least occurrence. It ensures that in the worst case, ECs generated from the bucket have the maximum change in probability for the values picked from the bucket, less than or equal to its upper bound.

The upper bound (u) for the least frequent value is calculated as:
$u = U(min\{p_b, p_{b+1}, p_{b+2}, \dots, p_e\})$; where $[b, \dots, e]$ represent the left to right sequence of values grouped together. The frequency of the SA value p_{e+1} in a subset that increases the sum of probabilities of values $\leq p_{e+1}$ over its upper bound, is split so that percentage of its frequency is added to the prior bucket. This brings the sum of probabilities in the bucket closer to the upper bound while still remaining less than or equal to it. The frequency required to create the overlapping bucket can be determined as: $x_l = \lfloor u_l \times \sum F \rfloor$, F being the overall frequency histogram of T and $\lfloor \dots \rfloor$ is a floor function to round the result to the nearest integer smaller than itself. Let, $y_l = \sum_{i=b}^{e} f_i$, then the required number of frequencies from f_{e+1} will be $x_l - y_l$.

The same process is performed for the remaining frequencies in the subset. If any values remain in a subset that could not be grouped into buckets with other values, the process is continued for subsets of values in the level parent to the current. Each bucket ensures a worse case upper bound so that an EC created with the least frequent values have a maximum percentage change of $min(\beta, -ln(p_i))$.

Taxonomy trees for string SA values are user defined. On the other hand, for numerical SA values, the taxonomy tree can be dynamically determined. Since numeric SA values also define observations/classifications, the variance in their values is limited and the size of the set of distinct values is small. Ninghui Li, et al. [5] and Jianneng Cao et al. [7] grouped SA values as a kd-tree through an iterative split function. They grouped the values into two subsets in each iteration. A value that minimized the distance between the distribution of each group and the original data set was chosen as the split value. The process continued to split the subsets further until the difference of their distribution was greater than a defined threshold.

ID	Zip	Age	Fav. Col.
1	47677	39	FF0000
2	47602	32	FF00F3
3	47678	37	FF00AF
4	47905	53	FF00F3
5	47909	62	0004FF
6	47906	57	00ABFF
7	47605	40	00ABFF
8	47673	46	00FFFF
9	47607	42	FF00AF

TABLE I: Data set T, with QIs=ZIP, Age & SA=$Fav.Col.$

Example 1: Bucketization: Let table I be the complete data set that has to be anonymized with columns $\{ID\}$ as identifier, $\{Zip, Age\}$ the QIs and $\{Fav.Col.\}$ as the SA. The anonymization process starts by bucketing the SA values. The taxonomy tree for Fav. Col. is shown in figure 1. The

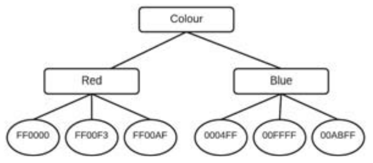

Fig. 1: Taxonomy tree of SA Fav. Colour

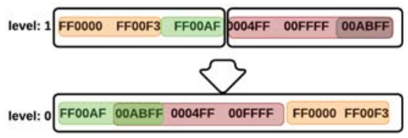

Fig. 2: Bucketization of SA Fav. Colour

taxonomy tree has a height $H = 2$ with level $l = 0$ representing the root node and level $l = 2$ the leaf nodes. The bucketization phase starts at level $l = H - 1$. It respectively measures the frequency histogram and the probabilities for all the SA values in all subsets at that level. The frequency histogram $F = \{\{1, 2, 2\}, \{1, 2, 1\}\}$ and probabilities $P = \{\{\frac{1}{9}, \frac{2}{9}, \frac{2}{9}\}, \{\frac{1}{9}, \frac{2}{9}, \frac{1}{9}\}\}$ for the SA values in table I at level $l = 1$, are grouped into two subsets as per their parent in the taxonomy tree. The bucketization phase, as shown in figure 2, sorts the frequencies and probabilities in each subset and creates the groups of SA values from each subset. With a user defined $\beta = 1.2$, the upper bounds for the two subsets are $U = \{0.35, 0.35\}$. In the first subset, grouping colors $FF0000, FF00F3$ bring their total probability to 0.33. While, if we try to include $FF00AF$ into this group, the probability increases to 0.56. In order to bring the probability of the first group in this subset closer to its upper bound, we need at least $0.35 \times 9 = 3.15$ or just frequency 3 (flooring the value to the nearest integer so that it never goes beyond the upper bound). The value $FF00AF$ cannot be grouped with any other value in this subset at this level. For the second subset, grouping values $0004FF, 00ABFF$ bring its probability to 0.22. We need at least 3 records to bring its probability closest to its upper bound. Since the total frequency of the group is 2, we can take $3 - 2 = 1$ frequency from the SA value $00FFFF$. This phase creates four groups of SA values $\{\{\{FF0000, FF0F3\}, \{FF00AF\}\}, \{\{0004FF, 00FFFF, 00ABFF\}, \{00ABFF\}\}\}$. These groups have frequencies and upper bounds respectively of $F_1 = \{\{3, 2\}, \{3, 1\}\}$ and $U = \{\{0.35, 0.56\}, \{0.35, 0.35\}\}$ (based on the least frequent value in each group). Next, we move to level $l = 0$ and group values $\{FF00AF, 00ABFF\}$ with respective frequencies $\{2, 1\}$ together bring their total probability to 0.33 which is smaller than their upper bound 0.35. This makes the groups complete, with maximum allowed frequency for each group being 3. The generated groups are complete with SAs $\{\{FF0000, FF0F3\}, \{FF00AF, 00ABFF\}, \{0004FF, 00FFFF, 00ABFF\}\}$. They can be represented as the final buckets having frequencies $\{3, 3, 3\}$ and upper bounds as $\{0.35, 0.35, 0.35\}$.

B. Dichotomization

Once the buckets are generated, records can be redistributed into ECs, as shown in example 2. The number of ECs and their sizes can be dynamically determined by dichotomizing or dividing the buckets in a top down approach. A binary tree is created based on sizes of the buckets, where each branch of the tree represents buckets divided approximately into two halves. The branches of the tree are recursively further divided into halves until the new probabilities of elements in each node of the branch remains less than equal to it upper bound in the root node. The leaf nodes of this tree are used as the number of ECs that can be generated from the set of buckets. The sizes determine the number of records that can be allocated to them.

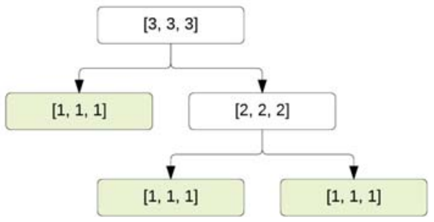

Fig. 3: Dichotomization of SA buckets

Example 2: Dichotomizing: Continuing from example 1, the dichotomization phase determines the number of ECs and their sizes as shown in figure 3. The three buckets from earlier are validated to determine whether dichotomizing them fulfills the upper bound requirement. The root buckets have a frequency histogram of $\{3, 3, 3\}$ and upper bound of $\{0.35, 0.35, 0.35\}$. If they are split into two child ECs, one with frequencies $\{1, 1, 1\}$ and another with $\{2, 2, 2\}$, the individual probability of each element of a bucket in the ECs is $\frac{2}{6} = \frac{1}{3} = 0.33$. Since the probabilities for all of the elements are smaller than their respective upper bounds, the split is allowed. The child with frequencies $\{1, 1, 1\}$ cannot be further dichotomized since doing so would recursively produce the same EC. The second EC with frequencies $\{2, 2, 2\}$ can be further split into two child ECs with frequencies $\{1, 1, 1\}$. Since the probability of each element in both ECs is 0.33 and is smaller than their respective upper bounds, the split is allowed. Finally, we have 3 ECs each having one record from each of the buckets.

C. Redistribution

The ECs can now be generated by picking records from each bucket such that it minimizes their information loss and increases the closeness of their distribution to the original data set. From each bucket, a set of initial seeds on its QIs for the ECs are determined based on KMeans++ [16]. It is represented in example 3.

Given the initial set of seeds representing the center of each EC for a bucket, the records in table T that have SA values belonging to the bucket are weighted based on their position on the taxonomy tree. Since the buckets are overlapping, the weights are based on the presence of SA values from the same parent from each bucket. Records from the same parent are weighted favorably in their distance measure from a given seed. This enables the created ECs to maintain the closeness of SA values to the distribution of the original table. It also ensures that the information loss is minimized within each ECs with regard to their QIs. Once the ECs are generated, they can be anonymized with the use of specific recording functions, as described in [15], [17], [18].

Example 3: Redistribution & Anonymization: Continuing from example 2, records from data set T, as shown in table I, are redistributed into the created ECs. The three ECs and their sizes determined from dichotomization phase are filled with records in order to minimize euclidean distance between QI values for records in an EC . Once the records are redistributed into ECs, the QIs in each EC are generalized/suppressed so that the values of QIs in records in an EC are identical to each other. The personal identifier values that uniquely identify a record are hashed. The anonymized data set is shown in table II. It represents the anonymized data set at two levels, first, having the SA values at the leaf level and, the second, showing the SA values at the parent level (as per their taxonomy tree). This demonstrates that the percentage increase in the probabilities of SA values are bounded by their upper bound, irrespective of the level of SA values.

In the anonymized table, three ECs were created having records with IDs $\{EC_1 = [4, 6, 9], EC_2 = [1, 5, 9], EC_3 = [2, 8, 3]\}$. The upper bound for SA value $FF0000$ in table T with probability ($p_i = 0.11$) is 0.35. The record with SA value $FF0000$ is redistributed into EC_2 where its SA probability remains less than the upper bound at $q_i = 0.33$. Even if the SA value is viewed from the point of view of its parent (Red), the upper bound constrain is still satisfied. The probability of Red ($p_i = 0.56$) in table T and appearing in the EC_2 with probability ($q_i = 0.67$) remains less that its upper bound (0.88).

ID	Zip	Age	Fav. Col.
#	47*	42-53	FF00F3
#	47*	42-53	00ABFF
#	47*	42-53	00ABFF
#	47*	39-62	FF0000
#	47*	39-62	0004FF
#	47*	39-62	FF00AF
#	476*	32-46	FF00F3
#	476*	32-46	00FFFF
#	476*	32-46	FF00AF

(a) level=leaf

ID	Zip	Age	Fav. Col.
#	47*	42-53	Red
#	47*	42-53	Blue
#	47*	42-53	Blue
#	47*	39-62	Red
#	47*	39-62	Blue
#	47*	39-62	Red
#	476*	32-46	Red
#	476*	32-46	Blue
#	476*	32-46	Red

(b) level=parent

TABLE II: Redistribution & Anonymization of data set T

D. Algorithms

This section introduces the algorithms for different phases of the Incognito solution. Algorithm 1 describes the general work flow of the solution. Step 1 creates the frequency histogram for all SA values and measures their probabilities through step 2. Step 3 creates the buckets with similar SA values and step

4 dynamically determines the number of ECs and their sizes. Step 5 redistributes the records from the buckets to create the required ECs.

Algorithm 1 workflow(T, β)

```
1: $F$ <- be the frequencies of distinct $SA$ values in data set $T$
2: $P$ <- generate the probability distribution of $F$
3: $(\phi, U)$ <- GenerateBuckets($F$, $P$, $level = H - 1$, $\beta$)
4: $ECSizes$ <- ECSizes($\phi$, $U$)
5: $ECs$ <- CreateEC($T$, $\phi$, $ECSizes$, $Size(ECSizes)$)
6: generalize the ECs to anonymize them
```

Algorithm 2, creates the overlapping buckets. Steps 1-3 define a default stop condition. Steps 4 & 5 create an empty vector ϕ and $\check{\phi} p$ that would respectively contain the frequencies of the buckets and the probability of the least frequent value in each bucket. Step 6 defines a variable that keeps count of the total number of buckets. Steps 6-32 create the intermediate buckets from the sub-sets (according to the taxonomy tree) of SA values. For each subset at a given level l, it sorts the distribution in that level through step 8. Step 9 defines a variable to measure the total probability of values in a bucket. It creates an initial upper bound value as positive infinity in step 10. Steps 11-32 determine the upper bound for the buckets that can be created from the subset and then creates a set of overlapping buckets. Each value in the subset is iterated over in a sequential manner and steps 12-15 ensure that the upper bound remains as the least frequent SA value among all the values that can be grouped into a single bucket. Step 16 updates the sum of a bucket with the probability of the SA value. Steps 17-30 check whether the added SA value would increase the probability of the bucket above its upper bound. It adds percentage of frequency of the SA value to the bucket that would bring its distribution closer or equal to its upper bound through step 18. It measures the percentage of frequencies of the total distribution to determine the number of SA values that can be grouped into the bucket. It then subtracts the total frequencies in the bucket from it as δ to determine the number of frequencies from the SA value that could be added to the bucket. Step 19 measures the probability of the δ as δ_p. If the upper bound of δ_p decreases the upper bound of the bucket, the amount of frequencies that can be added to the bucket is set to 0, in steps 20-22. The rest of the frequencies are used to create a new bucket with a new upper bound and total. Step 23 updates the frequency of bucket and step 24 increments the bucket counter by one and creates a new bucket to group this SA value with others. Step 25 updates the frequency of the SA value as the remaining number of frequencies after part of it is added to the previous bucket. Step 26 determines the probability of the remaining frequencies of the SA value. The total probability counter for the new bucket is set as the remaining probability through step 27 and its upper bound is calculated in step 28. Step 29 updates the minimum probability of the bucket. Step 31 updates the frequency of the bucket and step 33 re-iterates through the subsets prior to the current level in the taxonomy tree.

Algorithm 2 GenerateBuckets(F, P, l, β)

```
1: if $l < 0$ then
2:     return $(F, P \times (1 + min(\beta, -1 * ln(P)))$
3: end if
4: $\phi$ <- $NULL$; the set of buckets
5: $\check{\phi} p$ <- $NULL$; the probability of the least frequent value in a bucket
6: $n$ <- 0; number of buckets
7: for $\forall S_{lj} \subset F; j = 1, 2, \ldots, s_l$ do; $s_l$ being the number of sub-tree at level $l$
8:     Sort the distribution in $S_{lj}$
9:     $sum$ <- 0
10:    $u_n$ <- $+\infty$
11:    for $i = 1, 2, \ldots, |S_{lj}|$ do
12:        if $P_{lji} \times (1 + min(\beta, -1 * ln(P_lji))) < u_n$ then
13:            $u_n$ <- $P_{lji} \times (1 + min(\beta, -1 * ln(P_lji)))$
14:            $\check{\phi} p_{ljn}$ <- $P_{lji}$
15:        end if
16:        $sum$ <- $sum + P_{lji}$
17:        if $sum > u_n$ then
18:            $\delta$ <- $(\lfloor u_n \times \sum F \rfloor) - \phi_{ljn}$
19:            $\delta_p$ <- $\frac{\delta}{\sum F}$
20:            if $\delta_p \times (1 + min(\beta, -1 * ln(\delta_p))) < u_n$ then
21:                $\delta = 0$
22:            end if
23:            $\phi_{ljn}$ <- $\phi_{ljn} + \delta$
24:            $n$ <- $n + 1$
25:            $F_{lji}$ <- $F_{lji} - \delta$
26:            $P_{lji}$ <- $\frac{F_{lji}}{\sum F}$
27:            $sum$ <- $P_{lji}$
28:            $u_n$ <- $p \times (1 + min(\beta, -1 * ln(p)))$
29:            $\check{\phi} p_{ljn}$ <- $P_{lji}$
30:        end if
31:        $\phi_{ljn}$ <- $\phi_{ljn} + F_{lji}$
32:    end for
33:    $\phi$ <- GenerateBuckets($\phi$, $\check{\phi} p$, $l - 1$, $\beta$)
34: end for
35: return ($\phi$, $u$)
```

Algorithm 3 dynamically determines the sizes for ECs for a given list of bucket frequencies and the upper bounds of each bucket. Steps 2-4 create two child ECs, each containing approximately half of the SA frequencies. Steps 5-7 evaluate whether, for each of the ECs, the probability for the SA values in that EC goes beyond the upper bound. If the SA values in both ECs satisfy their upper bound, then these ECs are generated and further divided until they fail to satisfy their upper bound criteria through steps 8-13. The final number and sizes of ECs are determined as those that couldn't be further dichotomized.

Algorithm 4 creates the actual ECs after their sizes have been determined through Algorithm 3. It redistributes the

Algorithm 3 ECSizes(ϕ, U)

```
1: $EC_s$ <- $NULL$; Sizes of the ECs
2: $L$ <- $round(0.5 * \phi)$
3: $P_l$ <- generate the probability distribution of $L$
4: $R$ <- $\phi - L$
5: $P_r$ <- generate the probability distribution of $R$
6: $D1$ <- $p_i \in P_l \leq U_i; i = 1, 2, \ldots, |U|$
7: $D2$ <- $p_i \in P_r \leq U_i; i = 1, 2, \ldots, |U|$
8: if $\forall D1 == TRUE \wedge \forall D2 == TRUE$ then
9:     $EC_s$ <- $EC_s \cup$ ECSizes($L$, $U$)
10:    $EC_s$ <- $EC_s \cup$ ECSizes($R$, $U$)
11: else
12:    $EC_s$ <- $\phi$
13: end if
14: return $EC_s$
```

records into the ECs based on their nearest neighbors. Step 1 standardizes the values for all the QIs. Step 2 & 3 creates an empty list respectively for the seeds for each EC and the final ECs. Steps 4-12 redistribute the records from table T into the ECs. For each bucket, steps 5-10 re-allocate to records into ECs. Step 6 chooses points from the bucket as seeds for the ECs as described in algorithm 5. Step 7 weights the records in T whose SA values are present in the bucket as per the distance of their SA values to the SA value of the seed for an EC. It gives higher weights to records whose SA values are further from the seed of the EC and lower to the ones closer. Step 8 picks the required number of records for an EC from the buckets in order to minimize the weighted distance of their QIs. Since the buckets are overlapping, the penalties allow for choosing records from a bucket that have minimal QIs distance as well as are more similar in their SA values. Step 8 removes the records chosen to create the EC from T and the process re-iterates for the rest of the ECs and the remaining buckets.

Algorithm 4 CreateEC(T, ϕ, $ECSizes$)

```
1: Standardize the QIs of records in $T$
2: $seeds$ <- $NULL$;
3: $ECs$ <- $NULL$;
4: for $j$ in $1 : \phi$ do
5:     for $i$ in $1 : |ECSizes|$ do
6:         ($i$,$seeds_j$) <- InitialECSeeds($\forall t.QI \in T \wedge t.SA \in \phi_j, \#ofECs$)
7:         Weight the records $t$ in $T \wedge t.SA \in \phi_j$ as $\frac{heightToCommonParent(t_k.SA,(i,seeds_j.SA))}{height(T)}$
8:         $ECs_i$ <- $ECs_i \cup$ Pick the $ECSizes_{ij}$ number of records $t \subset T$ whose SA values are in $\phi_j$ so that it minimizes the weighted euclidean distance based on their QIs with center as ($i$, $seeds_j$)
9:         $T$ <- $T - ECs_i$
10:    end for
11: end for
12: return ECs
```

Algorithm 5 determines the initial centers for each EC. It is provided with the records from T with SA belonging to a bucket and the number of seeds to be chosen form the bucket. Based on concepts from KMeans++ [16], it determines the centers for each EC. Step 1 creates an oversampling factor. Step 2 selects a data point at random as the first center C. Step 3 determines the overall clustering cost of its QIs and the whole data set. Bahman Bahmani et al. proposed scalable k-means++ algorithm [19] to reduce the number of iterations. It uses an oversampling method to drastically reduce the iteration rounds from k to approximately $O(log\psi)$. Accordingly, the oversampling method is used and through steps 4-6 multiple points are sampled that are further away from each other. Step 5 samples each point t whose SA value is not present in the set of already chosen centers C independently with probability $l \times \frac{Diversity\ of(t,C)}{Clustering\ Cost\ of\ (C,T)}$. After $O(log(\psi))$ iterations, the number of chosen points is higher than k. Steps 7 & 8 use a weighted k-means++ clustering to obtain the final k centers.

Algorithm 5 InitialECSeeds(T, k)

```
1: $l$ <- $2 \times k$ as oversampling factor
2: $C$ <- sample a point uniformly at random from $T$
3: $\psi$ <- Clustering Cost of $(C, T)$
4: for $i$ in $1 : log(\psi)$ do
5:     $C'$ <- sample each point $t \subset T \wedge (\forall_{c \in C}\ h(t.SA, c.SA)\ != 1 || H(T) == 1)$ independently
6:     $C$ <- $C \cup C'$
7: end for
8: $\forall c \in C$, set $w_c$ to be the number of points in $C$ closer to $c$ then any other points in $C$
9: Re cluster the weighted points into $k$ clusters
10: $C$ <- centroid of each cluster
11: return $C$
```

IV. Empirical Evaluation

In this section Incognito the framework for large scale data anonymization in the cloud is evaluated using Incognito, t-closeness [7] and β-likeness [8] algorithms. The goal of the evaluation is to benchmark the framework for performance and scalability. The Incognito algorithm is also measured for sensitivity, identifiability and utility.

The ISLR Wage data set [20] was used for all experiments. The SA attribute $\{Wage\}$ was used as label and $\{yr, age, maritl, race, edu, jobclass, health, health\_ins\}$ QI attributes were used as features. A user defined taxonomy tree for the SA attribute was created with 6 levels as represented in figure 4. Level 5 represents the actual unique salaries present in the data set and at each prior level a parent was defined. This data set of 3000 records was also used as seeds to generate larger data sets of sizes up to 100 GigaBytes. Each record was also concatenated with a unique identifier as its Personal Identifier.

The implementations of the algorithms discussed in section III-D were made in a scalable manner using Apache Spark [21] version 1.6.0. The Incognito framework furnishes

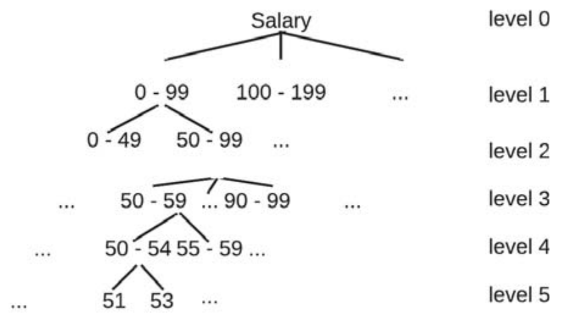

Fig. 4: Taxonomy of SA values Wage (in 1000 USD) with 6 levels

the bucketization for β-likeness and t-closeness as respectively described in [8] and [7]. The framework also allows dichotomization and redistribution for all algorithms as shown in sections III-B and III-C.

An execution environment of 12 CentOS Linux machines, each with 4 cores and 20 GB ECC DDR-2 RAM, was available for the experiments. An Apache Hadoop 2.6.2 file system (HDFS) and Apache Spark 1.6.0 were deployed on these machines. Both HDFS namenode and Spark master were deployed to the same machine. The remaining 11 machines were used as data nodes for HDFS and slave nodes for Spark.

A. *Scalability*

The scalability tests were performed in two folds. First, a data set of $\{1GB, 10GB, 20GB, \ldots, 100GB\}$ was generated from the ISLR Wage data set and stored into the HDFS file system. The algorithms using the framework were executed and evaluated over each data set, using 11 slave machines. Next, for the second set of scalability tests, a data set of 5GB was generated from the ISLR wage data and the framework was again evaluated on this data set, but the number of worker machines available for each test varied as $\{1, 3, 5, \ldots, 11\}$. Further, for both of these sets of experiments, the threshold value t and β for these algorithms was set so that the number of generated ECs by them was close to each other.

1) Data Sizes: The scalability tests for the framework were conducted using 11 machines and a varied set of data with different sizes from $1GB$ to $100GB$. The three algorithms t-closeness, β-likeness and Incognito were evaluated for their performance in terms of execution time (in seconds). The four phases of these algorithms used to anonymize a data set and the their overall execution time are shown in figure 5 with x-axis showing the varied data sizes and y-axis representing the execution time.

The first phase in anonymization is the Buckatization phase is shown in the graph (a). For all three algorithms, the execution time is almost equivalent and there is a linear growth in execution time, proportional to the growth in the data sizes. Graph (b) represents the dichotomization phase. t-closeness performs best in this phase since the number of buckets generated in the earlier phase remains constant based on the taxonomy tree of the SA values. This leads to a lower number of iterations for dichotomizing these buckets and determining the ECs. β-likeness and Incognito have similar linear performance in this phase, with Incognito performing slightly better with larger data sizes The redistribution phase, as shown in graph (c), parallelizes the redistribution based on number of generated buckets. t-closeness always generates a smaller number of buckets and its scalability suffers with larger data sets. β-likeness also suffers due to re-shuffling of records over the network between multiple machines as its buckets contain dis-similar records and for each generated EC, records have to be reshuffled. Incognito performs better for larger data sets as its generated buckets have similar records as patterns of QIs that lead to similar SA labels are grouped together. Graph (d) represents the recoding through generalization, suppression or summary statistics on QIs in each EC. They scale based on the number of generated ECs and, with all three algorithms generating almost equal number of EC, their performance is similar for smaller data sets. However, like the redistribution phase, it also suffers from reshuffling of records more for t-closeness and β-likeness than Incognito.

As seen in graph (e), the Incognito framework allows all three algorithms to scale. Furthermore, for smaller data sets, all three methods perform similarly. However, as the data size grows, our Incognito anonymization technique gradually outperforms the rest, having average of 75% and 35% better performance respectively against t-closeness and β-likeness.

2) Machines: The next set of experiments verifies the scalability of the framework by having a constant data size of $5GB$. The anonymization methods on this data set were executed by varying/scaling the number of machines from 1 to 11. Figure 6 shows the number of available machines on the x-axis and the overall execution time (in seconds) on the y-axis. The framework scales the algorithms as the number of available worker machines increases. The plateau in performance after 7 machines is due to the inherent nature of distributed computing where each computation is performed on a block of a particular size. Whenever machines used to handle the number of blocks for the $5GB$ of data are available, any additional computing resource does not significantly influence the performance. However, it does show the scalability of the framework in terms of addition of more computing resources through worker machines.

As seen from subsection IV-A1, t-closeness performs better for smaller data sets as the number of buckets generated in its buckatization phase is easily handled by the available computing resource. Both β-likeness and Incognito display similar performance for smaller data sets as the amount of re-shuffling of records over the network is limited.

B. *Identifiability*

The identifiability experiments evaluate the probability of identifying a sensitive attribute appearing in an anonymized group and is represented in figure 7. The x-axis represents the level of observation on the sensitive attribute. Level 5

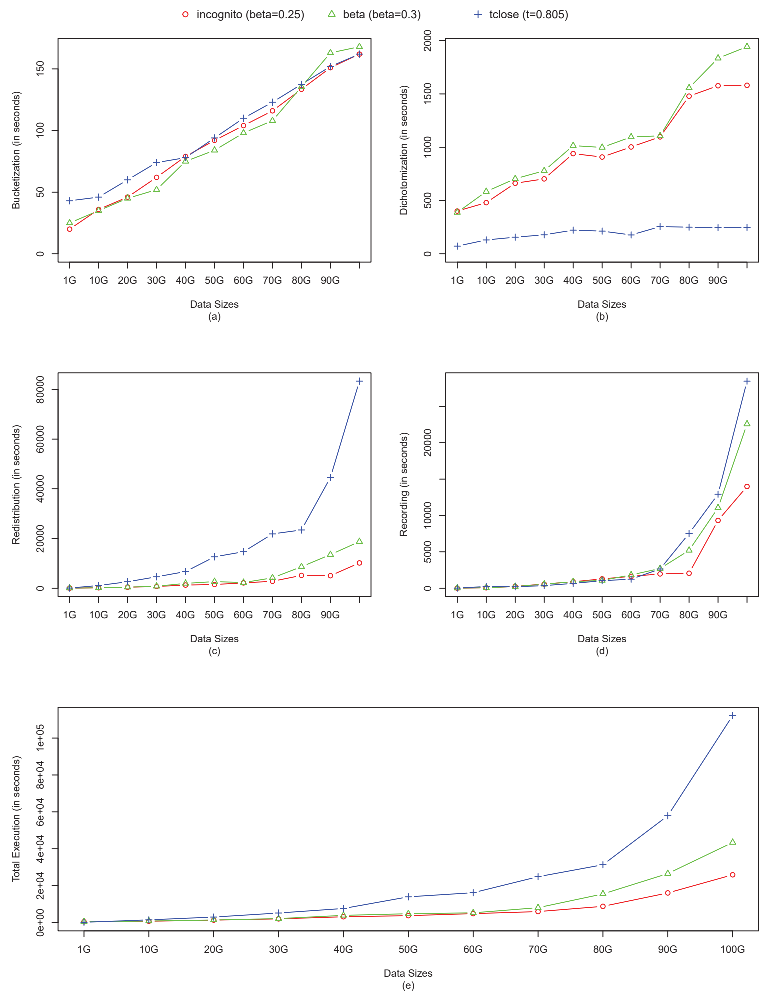

Fig. 5: Scalability using 11 machines and varying data sizes

shows the identifiability when sensitive attributes are used to identify without any prior knowledge. Level 4 represents prior knowledge about semantic similarities in terms of parents of the leaf node sensitive attribute values. Level 3 shows parents for values on Level 4 and so on.

As observed, at all level Incognito has better ability to decrease the average probability of SA values appearing in an EC. t-closeness performs worse at level 5 as it never limits the probability change of SA values appearing in an EC. Both β-likeness and Incognito limit the probability growth with the same threshold β=3.0 or 300%. However, β-likeness is not bounded to a taxonomy tree and it therefore moves closer to the upper bound. As we move to the parent levels, t-closeness and Incognito are able to show similar limits on the probability as both of them adhere to the taxonomy tree of the SA values. Incognito performs on average, over all levels, better than t-closeness and β-likeness (40% and 45% respectively) in hiding the identifiability of individuals, based on their SA values appearing in ECs of an anonymized data set.

C. *Utility*

The utility of the anonymized data sets was determined using different analytics functions. The QIs of the ISLR Wage data set with 3000 records were used as features and SA values as label. The original and anonymized data sets were used for linear regression, decision trees, knn regression, neural network and random forest for comparing the Incognito algorithm against β-likeness and t-closeness anonymization algorithms.

As observed from figure 8, the utility of the anonymized data depends on the data structure, the analytical needs and the type of used anonymization technique. The x-axis of the graph represents the different analytic techniques and y-axis shows the Mean Absolute Percentage Error (MAPE). Each of these algorithms was executed on the original data set (without anonymization) and three anonymized data sets using respectively t-closeness, β-likeness and Incognito. As expected, the information loss is higher on analytics on the anonymized data. The Incognito method and beta-likeness

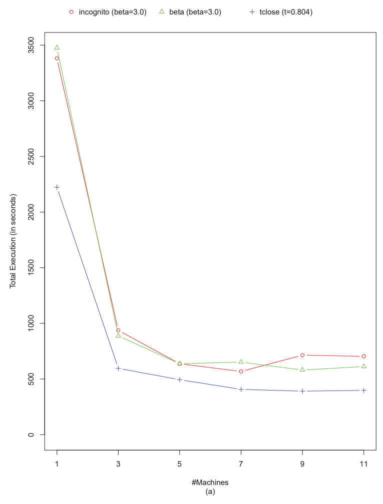

Fig. 6: Scalability using 5 GB of data and varying # of available worker machines

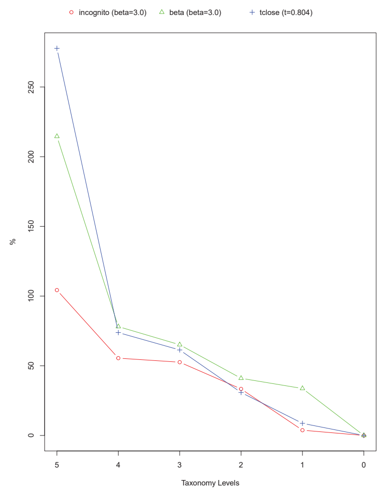

Fig. 7: Average % change of SA values in ECs compared to complete data set

performs similarly for most of the analytical functions while t-closeness performs the worst.

V. Related Work

l-diversity [6] & t-closeness [5] have been described as possible solutions to address some of the limitations of k-anonymization. Although l-diversity is able to well represent SA in an EC, it has been shown to suffer from semantic similarity and skewness attacks [5], [7]. t-closeness shows a promise in addressing the challenges in l-diversity by ensuring that the distance between distribution of a SA in an EC and the distribution of the attribute in the whole table is not higher than a threshold "t". A table is said to have t-closeness if all ECs have t-closeness. Still, as pointed out in [5], t-closeness suffers from challenges to multiple SAs as their approach becomes complex with the increase in the number of SAs. Further, the relationship between the value t and information gain is ambiguous. For example, the distance between the two distributions $P_1 = (0.01, 0.99)$ and $Q_1 = (0.11, 0.89)$ measured using Earth Mover's Distance (EMD)[1] [22] is, $EMD = 0.1$ and between distributions $P_2 = (0.4, 0.6)$ and $Q_2 = (0.5, 0.5)$ is also $EMD = 0.1$. However, the change between the first pair is much more significant than the second one as the probability of the first value 0.01 in P_1 and 0.11 in Q_1 increases by 1000%, where as for the second pair 0.4 in P_2 and 0.5 in Q_2 increases by just 25%. β-likeness [8], ensuring that an adversary's confidence on the likelihood of a certain SA value does not increase, in relative difference terms, by more than a predefined "β" threshold. It provides an effective privacy guarantee beyond t-closeness. However, β-likeness also has its constrains as it assumes the distribution to be sorted. This leads to failure to comply with the t-closeness constrains and makes it vulnerable to similarity attacks due to the fact that β-likeness ignores semantic hierarchical taxonomy trees of SA values.

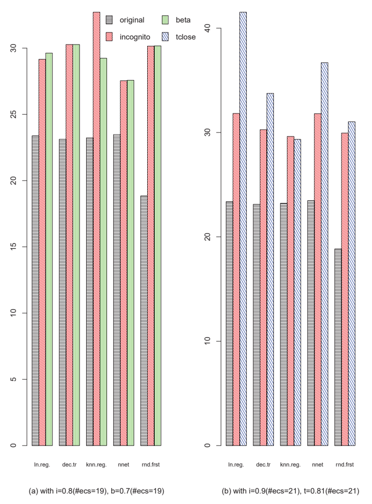

Fig. 8: MAPE of various learning algorithms on original and anonymized data (with similar # ECs)

VI. Conclusion

Data privacy has emerged as a key concern in today's data enabled services. Sharing and publication of micro data have lead to disclosure of sensitive private information about individuals. Popular privacy preserving techniques through anonymization have been shown to suffer from various attacks. Many alternative approaches to privacy have been also shown to suffer from some trade-offs. Further, anonymization techniques lack a framework for scalability in order to handle the large amounts of data generated across domains and industries.

[1]primary measure for calculating difference between two distributions in t-closeness

Though this paper we introduced a framework for scalable large scale data anonymization in the cloud. We also proposed our own anonymization algorithm called Incognito that addresses two of the major impediments towards anonymization, being similarity and skewness based attacks. β-likeness and t-closeness are two well accepted solutions that address these concerns to some extent. β-likeness provide a guarantee that probabilities of SAs in ECs are limited to a defined threshold. However, it suffers from similarity based attack since it does not adhere to the taxonomy tree of the SA values. t-closeness represents SA values into ECs as per their taxonomy tree. It generates ECs with SAs whose distribution is t close to their distribution in the original data set. However, it fails to represent individual probability increase of a SAs appearing in an EC and thus becomes liable to skewness based attacks. The proposed solution Incognito, address the limitations of both β-likeness and t-closeness.

Our evaluation of the framework and the algorithms showed that annonymization does not have one solution fit all scenario. Incognito is has shown better results in terms of performance for very large data sets. It also performs better when SA values in data sets have ontological similarity in terms of preventing identifiability especially when data sets have skewed SA distributions. Further, it also better limits the probability change of SA values appearing ECs when their SA taxonomy tree is known and the point of attacks are on the parent nodes. β-likeness suits data sets with SA values that are independent or numeric. t-closeness performs well for small data sets with ontologically related SA values that can be represented using a taxonomy tree. In terms of data utility or analytics, the most suitable anonymization method depends on the business requirements, data structures, data sizes, requirement of level of identifiability and the type of analytical function.

The research in the area of data privacy, anonymization and it usability is an ongoing process. In the future the framework would be further extended with other anonymization techniques including streaming anonymization. Additionally, various machine learning and analytical methods would be extensively evaluated on various data source to further determine relationships between type of anonymization and data utility.

References

[1] L. Sweeney, "k-anonymity: A model for protecting privacy," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 10, no. 05, pp. 557–570, 2002.

[2] B. Fung, K. Wang, R. Chen, and P. S. Yu, "Privacy-preserving data publishing: A survey of recent developments," *ACM Computing Surveys (CSUR)*, vol. 42, no. 4, p. 14, 2010.

[3] K. LeFevre, D. J. DeWitt, and R. Ramakrishnan, "Mondrian multidimensional k-anonymity," in *Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on.* IEEE, 2006, pp. 25–25.

[4] R. J. Bayardo and R. Agrawal, "Data privacy through optimal k-anonymization," in *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on.* IEEE, 2005, pp. 217–228.

[5] 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.* IEEE, 2007, pp. 106–115.

[6] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam, "l-diversity: Privacy beyond k-anonymity," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 1, no. 1, p. 3, 2007.

[7] J. Cao, P. Karras, P. Kalnis, and K.-L. Tan, "Sabre: a sensitive attribute bucketization and redistribution framework for t-closeness," *The VLDB Journal*, vol. 20, no. 1, pp. 59–81, 2011.

[8] J. Cao and P. Karras, "Publishing microdata with a robust privacy guarantee," *Proceedings of the VLDB Endowment*, vol. 5, no. 11, pp. 1388–1399, 2012.

[9] S. Chaudhuri, "What next?: a half-dozen data management research goals for big data and the cloud," in *Proceedings of the 31st symposium on Principles of Database Systems.* ACM, 2012, pp. 1–4.

[10] V. Borkar, M. J. Carey, and C. Li, "Inside big data management: ogres, onions, or parfaits?" in *Proceedings of the 15th International Conference on Extending Database Technology.* ACM, 2012, pp. 3–14.

[11] K. LeFevre, D. J. DeWitt, and R. Ramakrishnan, "Workload-aware anonymization techniques for large-scale datasets," *ACM Transactions on Database Systems (TODS)*, vol. 33, no. 3, p. 17, 2008.

[12] T. Iwuchukwu and J. F. Naughton, "K-anonymization as spatial indexing: Toward scalable and incremental anonymization," in *Proceedings of the 33rd international conference on Very large data bases.* VLDB Endowment, 2007, pp. 746–757.

[13] K. LeFevre and D. DeWitt, "Scalable anonymization algorithms for large data sets," *Age*, vol. 40, p. 40, 2007.

[14] X. Zhang, C. Liu, S. Nepal, C. Yang, W. Dou, and J. Chen, "Combining top-down and bottom-up: Scalable sub-tree anonymization over big data using mapreduce on cloud," in *Trust, Security and Privacy in Computing and Communications (TrustCom), 2013 12th IEEE International Conference on.* IEEE, 2013, pp. 501–508.

[15] L. Sweeney, "Achieving k-anonymity privacy protection using generalization and suppression," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 10, no. 05, pp. 571–588, 2002.

[16] D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding," in *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms.* Society for Industrial and Applied Mathematics, 2007, pp. 1027–1035.

[17] P. Samarati and L. Sweeney, "Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression," Technical report, SRI International, Tech. Rep., 1998.

[18] V. S. Iyengar, "Transforming data to satisfy privacy constraints," in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM, 2002, pp. 279–288.

[19] B. Bahmani, B. Moseley, A. Vattani, R. Kumar, and S. Vassilvitskii, "Scalable k-means++," *Proceedings of the VLDB Endowment*, vol. 5, no. 7, pp. 622–633, 2012.

[20] G. James, D. Witten, T. Hastie, and R. Tibshirani, "Package islr," p. 12, 2015. [Online]. Available: https://cran.r-project.org/web/packages/ISLR/ISLR.pdf

[21] A. Spark, "Apache spark home page, retrieved from internet on 23 september 2014," 2014.

[22] Y. Rubner, C. Tomasi, and L. J. Guibas, "The earth mover's distance as a metric for image retrieval," *International journal of computer vision*, vol. 40, no. 2, pp. 99–121, 2000.