Privatezza e Protezione dei Dati - Papers

Francesco Fontana, Alessandro Marchetti, Alfredo Santarcangelo

2024

0.1 Introduction

Chapter 1

k-anonimity

Contesto è il rilascio di *microdata*. De-identificazione non garantisce anonimità.

1.1 *k*-anonimity e Table *k*-anonime

Il concetto di k-anonimity cerca di catturare sulla Private Table (PT) il vincolo che i dati rilasciati dovrebbero essere associabili in maniera indistinguibile a non meno di un certo numero di respondent.
Il set di attributi disponibili esternamente e quindi sfruttabili per fare linking è chiamato *quasi-identifier*.

Definizione 1 (*k*-anonimity requirement)
Ogni rilascio di data deve essere tale che ogni combinazione di valori del Quasi-Identifier può essere matchata in maniera indistinguibile con almeno k respondent.

k-anonimity richiede che ogni valore del *Quasi-Identifier*abbia almeno k occorrenze nella table rilasciata, come da def 1.1 che segue:

Definizione 2 (*k*-anonimity)
Date una table $T(A_1, A_2, ...A_m)$ *e un insieme di attributi* QI*, Quasi-Identifier sulla table* T*:*
T *soddisfa k-anonimity rispetto a* QI *se e solo se ogni sequenza di valori in* $T[QI]$ *appare almeno con k occorrenze in* $T[QI]$[1]

Definizione 1.1 è sufficiente per k-anonimity. Applicazione di k-anonimity richiede una preliminare identificazione del *Quasi-Identifier*.

Il *Quasi-Identifier* dipende dalle informazioni esterne disponibili al recipiente poichè determina le capacità di linking dello stesso. Diversi *Quasi-Identifier* possono potenzialmente esistere per una data table.

Per semplicità a seguire nel paper si assume che:

[1] $T[QI]$ denota la proiezione con tuple duplicate degli attributi QI in T

- PT ha unico *Quasi-Identifier*.
- *Quasi-Identifier* è composto da tutti gli attributi nella PT disponibili esternamente.
- PT contiene al massimo una sola tupla per ogni respondent.

k-anonimity si concentra su due tecniche di protezione: *Generalization* e *Suppression*, le quali preservano la veridicità dei dati (diversamente da swapping e scrambling).

1.1.1 *Generalization*

Sostituzione dei valori di un attributo con valori più generali. Consideriamo:

- *Domain*: set di valori che un attributo può assumere.
- *Generalized domains*: contiene valori generalizzati e relativo mapping tra ogni domain e ogni sua generalizzazione.
- `Dom`: set di domini originali con le loro generalizzazioni.
- *Generalization relationship* $\leq_D$: dati D_i, $D_j \in$ `Dom`, $D_i \leq D_j$ significa che i valori in D_j sono generalizzazioni dei valori in D_i.

  $\leq_D$ definisce ordinamento parziale su `Dom` ed è richiesto nelle seguenti condizioni:

Condizione 1 (C1 - Determinismo nel processo di generalizzazione)
$\forall D_i, D_j, D_z$ *`Dom`:*
$D_i \leq_D D_j$, $D_i \leq_D D_z \implies D_j \leq_D D_z \vee D_z \leq_D D_j$[2].

Condizione 2 (C2 -)
Tutti gli elementi massimali di `Dom` *sono singoletti (singleton)* [3].

- $\mathbf{DGH}_D$ - *Domain Generalization Hierarchy*: gerarchia di ordninamento totale per ogni dominio $D \in$ `Dom`.

Per quanto riguarda i valori nei domini consideriamo:

- *Value generalization relationship* $\leq_V$: associa ogni valore in D_i ad un unico valore in D_j, sua generalizzazione.
- $\mathbf{VGH}_D$ - *Value Generalization Hierarchy*: albero dove
  - Foglie sono valori in D.
  - Radice è il valore, singolo, nell'elemento massimale di DGH_D

[2] Questo comporta che ogni dominio D_i ha al massimo un solo dominio di generalizzazione diretta D_j

[3] La condizione assicura che tutti i valori in ogni dominio possano essere generalizzati ad un singolo valore

1.1.2 *Suppression*

Consideriamo Soppressione di Tupla. "Modera" la *Generalization* quando un numero limitato di *outlier*[4] forzerebbe una generalizzazione elevata.

1.2 Generalizzazione *k*-Minima

Definizione 3 (Table Generalizzata con Soppressione)
Consideriamo T_i e T_j due table sugli stessi attributi.
T_j è generalizzazione (con soppressione di tupla) di T_i, riportata come $T_i \preceq T_j$, se:

1. $|T_j| \leq |T_i|$
2. *Dominio $dom(A, T_j)$ è uguale o una generalizzazione di $dom(A, T_i)$, dove A indica ogni attributo in $T_{i,j}$*
3. *E' possibile definire funzione iniettiva che associa ogni tupla $t_j \in T_j$ con una tupla $t_i \in T_i$, per la quale ogni attributo in t_j è uguale o generalizzazione del corrispondente in t_i.*

Definizione 4 (Distance Vector)
Siano $T_i(A_1, ..., A_n)$ e $T_j(A_1, ..., A_n)$ tali che $T_i \preceq T_j$.
il distance vector di T_j da T_i è il vettore
$DV_{i,j} = [d_1, ..., d_n]$
dove ogni $d_z, z = 1, ..., n$ è la lunghezza dell'unico percorso tra $dom(A_z, T_i)$ e $dom(A_z, T_j)$ nella DGH_{D_z}

Corollario 1 (Ordine Parziale tra DV)
$DV = [d_1, ..., d_n] \leq DV' = [d'_1, ..., d'_n]$ se e solo se $d_i \leq d'_i$ per $i = 1, ..., n$.

Si costruisce una *gerarchia di distance vectors* come lattice (diagramma) corrispondente alla DGH_D come in fig. 1.1

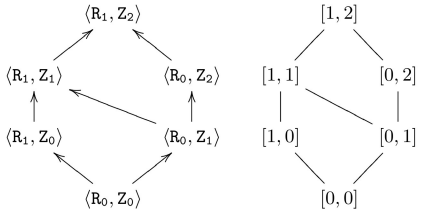

Figure 1.1

[4] TODO outlier

Per bilanciare tra perdita di precisione dovuta a *Generalization* e perdita di completezza dovuta a *Suppression* si suppone che data holder determini la soglia `MaxSup`, che indica il numero di tuple che possono essere soppresse.

Definizione 5 (Generalizzazione *k*-minima con Soppressione)
Siano T_i e T_j due table tali che $T_i \preceq T_j$, e sia `MaxSup` la soglia di soppressione accettabile scelt. T_j è una generalizzazione k-minima di T_i se e solo se:

1. *T_j soddisfa k-anonimity applicando soppressione minima, ossia T_j soddisfa k-anonimity e: $\forall T_z : T_i \preceq T_z$, $DV_{i,z} = DV_{i,j}$, T_z soddisfa k-anonimity $\implies |T_j| \geq |T_z|$.*
2. *$|T_i| - |T_j| \leq$ `MaxSup`.*
3. *$\forall T_z : T_i \preceq T_z$ e T_z soddisfa le condizioni 1 e 2 $\implies \neg(DV_{i,z} < DV_{i,j})$.*

Ultima espressione rende meglio come $DV_{i,z} \geq DV_{i,j}$. Il concetto che esprime è che *"non esiste un'altra Generalization T_z che soddisfi 1 e 2 con un DV minore di quello di T_j"*

Diversi **preference criteria** possono essere applicati nella scelta della generalizzazione minimale preferita:

- **Distanza assoluta minima**: minor numero totale di passi di generalizzazione (indipendentemente dalle gerarchie di *Generalization* considerate).
- **Distanza relativa minima**: minimizza il numero relativo di passi di generalizzazione (passo relativo ottenuto dividendo per l'altezza del dominio della gerarchia a cui si riferisce.
- **Massima distribuzione**: maggior numero di tuple distinte.
- **Minima soppressione**: minor tuple soppresse (maggior cardinalità).

1.3 Classificazione tecniche di *k*-anonimity

Classificazione in fig. 1.2.

	Suppression			
Generalization	*Tuple*	*Attribute*	*Cell*	*None*
Attribute	**AG_TS**	**AG_AS** ≡ AG_	**AG_CS**	**AG_** ≡ AG_AS
Cell	**CG_TS** not applicable	**CG_AS** not applicable	**CG_CS** ≡ CG_	**CG_** ≡ CG_CS
None	**_TS**	**_AS**	**_CS**	_ not interesting

Fig. 8. Classification of k-anonymity techniques

Figure 1.2

Casi *not applicable* (**CG_TS** e **CG_AS**): supportare *Generalization* a grana fine (cella) implica poter applicare soppressione allo stesso livello.

Algorithm	**Model**	**Algorithm's type**	**Time complexity**
Samarati [26]	AG_TS	Exact	exponential in $\|QI\|$
Sweeney [29]	AG_TS	Exact	exponential in $\|QI\|$
Bayardo-Agrawal [5]	AG_TS	Exact	exponential in $\|QI\|$
LeFevre-et-al. [20]	AG_TS	Exact	exponential in $\|QI\|$
Aggarwal-et-al. [2]	_CS	$O(k)$-Approximation	$O(kn^2)$
Meyerson-Williams [24][2]	_CS	$O(k \log k)$-Approximation	$O(n^{2k})$
Aggarwal-et-al. [3]	CG_	$O(k)$-Approximation	$O(kn^2)$
Iyengar [18]	AG_TS	Heuristic	limited number of iterations
Winkler [33]	AG_TS	Heuristic	limited number of iterations
Fung-Wang-Yu [12]	AG_	Heuristic	limited number of iterations

Figure 1.3: Alcuni approcci a k-anonimity(n è numero di tuple in PT).

1.4 Algoritmo Samarati (AG_TS)

Il primo algoritmo per garantire k-anonimity è stato proposto insieme alla definizione di k-anonimity. La definizione di k-anonimity è basata sul QI quindi l'algoritmo lavora solo su questo set di attributi e su table con più di k tuple.

Data una *DGH* ci sono diversi percorsi dall'elemento in fondo alla gerarchia alla radice. Ogni percorso è una differente *strategia* di generalizzazione. Su ogni percorso c'è esattamente una *Generalization* minima localmente (nodo più basso che garantisce k-anonimity).
In maniera naif si può cercare su ogni percorso il minimo locale per poi trovare il minimo globale tra questi ma non è praticabile per l'elevato numero di percorsi.

Per ottimizzare la ricerca si sfrutta la proprietà che salendo nella gerarchia la soppressione richiesta per avere k-anonimity diminuisce:

- Ogni nodo in DGH viene associato ad un numero, **height**, corrispondente alla somma degli elementi nel Distance Vector associato.
- Altezza di ogni DV nel diagramma (*distance vector lattice* `VL`) si scrive come $height(DV,\texttt{VL})$.

Se non c'è soluzione che soddisfi k-anonimity sopprimendo meno di `MaxSup` ad altezza h non può esistere soluzione che soddisfi ad una altezza minore.

L'algoritmo usa binary search cercando la minore altezza in cui esiste un DV che soddisfa k-anonimity rispettando `MaxSup` e ha come primo passo:

$$\text{Cerco ad altezza } \lfloor \frac{h}{2} \rfloor : \begin{cases} \text{trovo vettore che soddisfa } k\text{-anonimity} & \implies \lfloor \frac{h}{4} \rfloor \\ \text{altrimenti} & \implies \text{cerco in } \lfloor \frac{3h}{4} \rfloor \end{cases} \tag{1.1}$$

La ricerca prosegue fino a trovare l'altezza minore in cui esiste vettore che soddisfa k-anonimity con `MaxSup`.

1.4.1 Evitare il calcolo delle table generalizzate

Algoritmo richiederebbe il calcolo di tutte le table generalizzate. Per evitarlo introduciamo il concetto di DV tra tuple.

Definizione 6 (Distance Vector tra tuple - Antenato Comune)
Sia T una table.
Siano $x, y \in T$ due tuple tali che $x = \langle v'_1, ..., v'_n \rangle$ e $y = \langle v''_1, ..., v''_n \rangle$ con v'_i e v''_i valori in D_i con $i = 1, ..., n$.
*Il **distance vector** tra x e y è $V_{x,y} = [d_1, ..., d_n]$. dove d_i è la lunghezza (uguale) dei due percorsi da v'_i e v''_i al loro comune antenato comune più prossimo v_i sulla VGH_{D_i}.*

In altri termini ogni distanza in $V_{x,y}$ è una distanza uguale dal dominio di v'_i e v''_i al dominio in cui sono generalizzati allo stesso valore v_i.

Allo stesso modo $V_{x,y}$ per $x, y \in T_i$ equivale a $DV_{i,j}$ per $T_i \preceq T_j$ per cui x e y vengono generalizzate alla stessa tupla t.

Per il momento il resto è delirio

1.5 Bayardo-Agrawal: *k-Optimize* (AG_TS)

Approccio considera che la generalizzazione di attributo A su dominio **ordinato** D corrisponde ad un partizionamento del dominio dell'attributo in intervalli. Ogni valore del dominio deve comparire in un intervallo e ogni valore in un intervallo precede ogni valore degli intervalli che lo seguono.

Si assume quindi un ordine tra gli attributi del *Quasi-Identifier*. Inoltre associa un valore intero chiamato *index* ad ogni intervallo di ogni dominio degli attributi del *Quasi-Identifier*.

Una *Generalization* è quindi rappresentata come l'unione dei valori di indice per ogni attributo (il valore più piccolo in un dominio viene omesso perchè comparirà sicuramente nella generalizzazione per quel dominio).

k-Optimize costruisce un *set enumeration tree* sul set I di valori di indice, con radice vuota. Ogni nodo figlio è costruito dal padre inserendo in coda un index di I maggiore degli altri index già presenti nel padre (per ordinamento totale).

La visita dell'albero permette di valutare con Depth First Search ogni nodo, prunando ogni nodo e tutti i suoi figli se questi non possono corrispondere a soluzioni ottimali. Nello specifico *k-Optimize* pruna un nodo n quando determina che nessuno dei suoi discendenti può essere ottimale. Data una funzione di costo l'algoritmo calcola un limite inferiore sul costo che può essere ottenuto sul subtree del nodo n, prunando se nodo a costo minore è già stato trovato.

Se un nodo viene prunato allora anche altri nodi, non per forza del sottoalbero, possono essere prunati: supponendo di prunare il nodo {1,3} in figura 1.4 allora posso prunare qualunque altro nodo contenente 1 E 3, come ad esempio {1,2,3}.

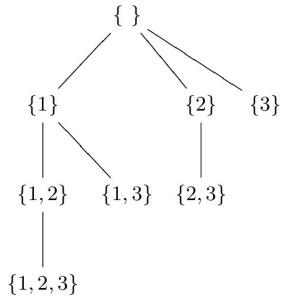

Figure 1.4: Set enumeration tree sull'insieme di indici I={1,2,3}

1.6 LeFevre-DeWitt et al.: Incognito (AG_TS)

Idea di base è riassunta nella seguente definizione:

Definizione 7 (k-anonimity dei subset di QI)
Se una table T con Quasi-Identifier QI composto da $m = |QI|$ attributi soddisfa k-anonimity, T soddisfa k-anonimity anch per qualunque Quasi-Identifier QI' talce che $QI' \subset QI$.

Pertanto k-anonimity su un subset di QI è condizione necessaria (e non sufficiente) per k-anonimity di T sull'intero QI.

Algoritmo esclude in anticipo alcune generalizzazioni della gerarchia con un calcolo a priori.

Strategia: bottom-up BFS sulla DGH. Incognito genera tutte le possibili table k-anonime minime secondo i seguenti passaggi:

1. (Iterazione 1): verifica k-anonimity per ogni singolo attributo nel *Quasi-Identifier*, scartando quelle che non soddisfano.
2. (Iterazione 2): combina a coppie le *Generalization* non scartate al passo 1 verificando k-anonimity.
3. (...)
4. (Iterazione m): arriva a considerare l'intero set di attributi di QI

Utilizzando approccio bottom-up, per la condizione citata prima, se una *Generalization* soddisfa k-anonimity allora anche sue ulteriori dirette generalizzazioni soddisfano k-anonimity e pertanto non sono considerate ulteriormente.

1.6.1 esempio Incognito

Chapter 2

Cloud Security: Issues and Concerns

2.1 Summary

Il presente paper si pone l'obiettivo di spiegare come il garantire la sicurezza significa anche assicurare la *confidenzialità*, *integrità* dei dati e anche la loro *disponibilità*, CIA in una parola.

2.2 Introduzione

Nella prima parte

2.3 CIA nel Cloud

2.4 Problemi e Sfide