IT Biztonság (VIHIAC01)
Házi feladat

Adatvédelem: Adatbázis-lekérdezések Auditálása

Szerző:
Ács Gergely

2022. április 14.

Tartalomjegyzék

1. Háttér

A lekérdezések auditálása a múltban megválaszolt adatbázis-lekérdezések vizsgálata annak megállapítására, hogy egy lekérdező (támadó) használhatta-e az ilyen lekérdezésekre adott válaszokat bizalmas információk megállapítására. Ennek a házi feladatnak a fókusza a statisztikai adatbázisokon végrehajtott lekérdezések auditálása lesz. Az adatbázis egyetlen privát attribútumot tartalmaz, amelyen csak `SUM` lekérdezést lehet végrehajtani.

1.1. Egy illusztratív példa

A probléma szemléltetéséhez vegyük a következő példát. A kórházi adatbázis az 1. táblázatban található, a privát (érzékeny) attribútum a `Blood sugar`. A lekérdezéseket egy `WHERE` feltétel (predikátum) és egy aggregálást végző függvény határoz meg. A feltétel kiválaszt néhány rekordot az adatbázisból, majd az aggregált függvény végrehajtásra kerül a kiválasztott rekordok egy (privát) attribútumán. A lekérdezés eredménye az aggregált függvény visszatérési értéke. Például a `SELECT SUM("Blood sugar") WHERE Gender = "Male"` lekérdezés eredménye 16.6. A lekérdezés predikátuma `Gender = "Male"`, és a kiválasztott rekordok (férfiak) rekordján kiszámított aggregált függvény a `SUM`.

#	Name	ZIP	Gender	Blood sugar
1	John Smith	32453	Male	4.3
2	Jeremy Doe	43813	Male	5.2
3	Susan Atkinson	43765	Female	6.1
4	Eve Brown	32187	Female	3.2
5	Joseph Sky	33745	Male	7.1
6	Alison Moon	22983	Female	6.2

1. táblázat. Kórházi adatbázis. Kék attribútumok publikusak és szerepelhetnek egy lekérdezés predikátumában (`WHERE` feltétel). A piros attribútumok érzékenyek és csak aggregált függvények értékelhetők ki rajtuk (pl. `SUM`), predikátumban nem szerepelhetnek

Nem nehéz belátni, hogy a következő kérdések megválaszolása felfedi a második rekord (Jeremy Doe) vércukorszintjét, annak ellenére, hogy minden lekérdezés legalább két rekordot kiválaszt.

- 1. lekérdezés: `SELECT SUM("Blood sugar") FROM Dataset`; Eredmény: 32.1
- 2. lekérdezés: `SELECT SUM("Blood sugar") FROM Dataset WHERE Gender = "Female"`; Eredmény: 15.5
- 3. lekérdezés: `SELECT SUM("Blood sugar") FROM Dataset WHERE ZIP > 32000 and ZIP < 35000 AND Gender = "Male"`; Eredmény: 11.4

Hogy ez miért igaz, vegyük észre, hogy az 1. és 2. lekérdezés visszaadja az összes férfi összesített vércukorszintjét (vonjuk ki a 2. lekérdezés eredményét az 1. lekérdezés eredményéből), és a 3. lekérdezés csak az 1. és az 5. rekordot választja ki. Ezért a 3. lekérdezés eredményét levonva és a férfiak összesített vércukorszintjének eredményéből megkapjuk a 2. rekord pontos vércukorszintjét.

Formálisabban, jelölje x_i az i-ik rekord ismeretlen (privát) vércukorértékét az 1. táblázatban. Minden lekérdezés és annak eredménye egy lineáris egyenlettel leírható az alábbiak szerint:

- 1. lekérdezés: $x_1 + x_2 + x_3 + x_4 + x_5 + x_6 = 32.1$
- 2. lekérdezés: $x_3 + x_4 + x_6 = 15.5$
- 3. lekérdezés: $x_1 + x_5 = 11.4$

Mátrixos formában:

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{1}$$

ahol

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$
$$\mathbf{x} = (x_1, x_2, \ldots, x_6)^\mathsf{T}$$
$$\mathbf{b} = (32.1, 15.5, 11.4)^\mathsf{T}$$

A feladat annak ellenőrzése, hogy az (1). egyenletrendszer egyértelműen megoldható-e *bármely* x_i-re. Ha igen, akkor az $\mathbf{A}$ mátrix által megadott lekérdezéseket nem lehet megválaszolni anélkül, hogy felfednénk a rekordok privát

attribútumának értékét. A következő részben először pontosabban meghatározzuk a lekérdezések off-line auditálásának problémáját, majd bemutatunk egy technikát, amellyel jól becsülhetjük $\mathbf{x}$ értéket a (1). egyenletrendszerben függetlenül attól, hogy hány lekérdezésünk van.

Mejegyzés. *A (1). egyenletrendszer x_2-ra történő megoldása látszólag nem jelenthet semmilyen adatvédelmi problémát a GDPR szerint, hiszen úgy tűnik nem tudjuk a visszaállított értéket egy fizikai személyhez (Jeremy Doe-hoz) társítani. A GDPR szerint minden információ személyes, ha azonosított vagy azonosítható személyhez köthető. Valóban, ha a támadó csak a nyilvános attribútumokhoz fér hozzá (kék színnel jelöltek), akkor valószínűleg nem tudja összekapcsolni x_2 értékét Jeremy Doe-val. Ugyanakkor a támadónak mindig lehet némi extra háttérismerete, amely segíthet Jeremy rekordjának azonosításában, még akkor is, ha nem fér hozzá a "Name" attribútumhoz. Például egy másik forrásból (pl. Facebook) lehet tudni, hogy Jeremy kórházba ment, és irányítószáma 438-cal kezdődik. Ha ilyen extra tudásnak a megszerzése "ésszerűen" reális feltételezés, akkor a fenti 1., 2. és 3. lekérdezések eredményei önmagukban személyesnek és érzékenynek tekinthetők még a GDPR szempontjából is.*

1.2. Lekérdezések Off-line Auditálása

Adott egy privát értékek halmaza $X = \{x_1, \ldots, x_n\}$. Az aggregált $q = (Q, f)$ lekérdezés a rekordok egy $Q \subseteq \{1, \ldots, n\}$ részhalmazát és egy f függvényt jelent, ami lehet például `SUM`. Az eredmény $f(Q)$ az f függvény alkalmazása a $\{x_i | i \in Q\}$ részhalmazra. A Q-t a q lekérdezési halmazának hívjuk. Legyen $\{b_1, \ldots, b_m\}$ a $\{q_1, \ldots, q_m\}$ aggregált lekérdezések eredményei X felett. A cél annak meghatározása, hogy egy rekord privát attribútum értéke kiszámolható-e a lekérdezésekből és azok eredményeiből. A (teljes) felfedés definíciója a következő.

1.1. Definition (Teljes felfedés)**.** Az $x_i \in X$ elemet egy Q lekérdezési halmaz teljesen felfed, ha x_i egyértelműen meghatározható, azaz minden lehetséges X adathalmazban ami összhangban áll a $q_1, \ldots, q_m$ lekérdezésekre adott $b_1, \ldots, b_m$ válaszokkal, x_i értéke azonos.

Általában véve f bármilyen lineáris vagy nem lineáris aggregált függvény lehet. A lineáris függvények közé tartozik a `SUM`, míg a nemlineáris függvények közé tartozik a `MEDIAN`, `MAX`, `MIN`, stb. Ha f nem lineáris, akkor a

teljes felfedés még polinom időben sem ellenőrizhető mindig [2]. Ilyen esetekben csak heurisztikához folyamodhatunk, például SAT megoldókat használhatunk [1]. Ha azonban f lineáris, akkor a teljes felfedés egy adott X adatbázis esetében egyszerűen ellenőrizhető az $\mathbf{Ax} = \mathbf{b}$ lineáris egyenletrendszer megoldásával, ahol $\mathbf{x} = X$, $\mathbf{b} = \{b_1, \ldots, b_m\}$, $\mathbf{A} \in \mathbb{R}^{m,n}$, ahol az $\mathbf{A}$ mártixban az i. sor reprezentálja a q_i lekérdezést. Ezt szemlélteti a fenti példa is, és ezzel az esettel fogunk csak foglalkozni ebben a házi feladatban.

Ha több lekérdezésünk van mint rekord ($m > n$), akkor egy túlhatározott lineáris egyenletrendszerrel kell szembenéznünk (ebben az esetben egyes lekérdezések szükségszerűen lineárisan függnek egymástól, mert az $\mathbf{A}$ mátrix rangja mindig kisebb mint n[1]). Ha több rekordunk van mint lekérdezés ($m < n$), akkor alulhatározott egyenletrendszerünk van gyakran végtelen számú megoldással. Az, hogy meg tudjuk-e oldani a $\mathbf{Ax} = \mathbf{b}$ egyenletrendszert bármely x_i esetén két dologtól függ: (1) $\mathbf{b}$ az $\mathbf{A}$ mátrix oszlopvektorainak a lineáris kombinációja-e, illetve (2) a lineárisan független lekérdezések számától, amely soha nem nagyobb, mint a rekordok száma, ami n. Mindkét követelmény az $\mathbf{A}^* = [\mathbf{A}|\mathbf{b}]$ kibővített mátrix rangjával (ami legfeljebb $n+1$) formalizálható az alábbiak szerint[2]:

- Ha $rank(\mathbf{A}) < rank(\mathbf{A}^*)$ (azaz $\mathbf{b}$ lineárisan független az $\mathbf{A}$ oszlopaitól), akkor az egyenletrendszernek nincs megoldása. Ha minden lekérdezést pontosan megválaszolnak, ez az eset nem állhat fenn.

- Ha $rank(\mathbf{A}^*) = rank(\mathbf{A}) = n$ (azaz $\mathbf{A}$ négyzetes mátrix és teljes rangja van), akkor a rendszernek egyértelmű megoldása van: $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$, ahol $\mathbf{A}^{-1}$ az $\mathbf{A}$ inverze.

- Ha $rank(\mathbf{A}^*) = rank(\mathbf{A}) < n$, akkor gyakran végtelen számú megoldás létezik, hacsak kifejezett korlátozások nem szűkítik le a megoldási teret, ami csak véges számú megoldást eredményez. Ilyen korlátozások közé tartozik, amikor az $\mathbf{x}$ egész számokból áll (például a mi esetünkben), vagy amikor $\mathbf{x}$ ritkás (sparse) (pl. a nullától eltérő koordinátaértékek száma legfeljebb $rank(\mathbf{A})$) [3].

[1] A lineárisan független sorok száma mindig megegyezik a lineárisan független oszlopok számával bármely mátrixban.

[2] Részletek: https://math.bme.hu/algebra/a2/2009/2_LinearisEgyenletrndszrk.002.pdf

[3] lásd MDS hibajavító kódokat vagy compressive sensing-et

Létezik néhány technika a lineáris egyenletrendszer megoldására, például Gauss-elimináció alkalmazható a kibővített mátrixon. Itt most egy egyszerűbb és rugalmasabb optimalizálási technikát fogunk használni, amely mindig képes egy becslést adni $\mathbf{x}$ értékére függetlenül $\mathbf{A}^*$ rangjától, és könnyen bővíthető bármilyen extra (lineáris) feltételekkel[4]:

$$\mathbf{x}' = \arg\min_{\mathbf{x}\in\mathbb{R}^n} \quad ||\mathbf{A}\mathbf{x} - \mathbf{b}||_2 \tag{2}$$

ahol $||\cdot||_2$ az euklideszi távolságot jelöli (L_2 norma). Ennek az optimalizálási problémának a megoldása zárt formában megadható: $\mathbf{x}' = \hat{\mathbf{A}}\mathbf{b}$, ahol $\hat{\mathbf{A}}$ az $\mathbf{A}$ mátrix *pszeudó inverze* (vagy Moore–Penrose inverze)[5]. Természetesen a $||\mathbf{x}' - \mathbf{x}||_2$ rekonstrukciós hiba függ az $\mathbf{A}$ mátrix rangjától (vagyis a lineárisan független lekérdezések számától), és nincs garancia arra, hogy $\mathbf{x}' = \mathbf{x}$, hacsak $\mathbf{A}$ nem rendelkezik teljes ranggal és $rank(\mathbf{A}) = rank(\mathbf{A}^*)$ a fentiek szerint[6]. A fenti optimalizációs technika *Ordinary Least Squares* (OLS) néven ismert[7], és számos területen használják.

Ebben a házi feladatban arra az esetre koncentrálunk, amikor (1) $\mathbf{x} \in \mathbb{Z}^n$, (2) f lineáris, és (3) $\mathbf{A}$ mátrix bináris, azaz $\mathbf{A} \in \{0,1\}^{m,n}$. Az (2). formula megoldása azonban valós értékű. Sajnos a mi esetünkben $\mathbf{x}$ egész értékű, amivel a fenti optimalizálási probléma egy egészértékű lineáris programozási feladattá módosul (Integer Liner Programming – ILP), amely NP-teljes. Ahelyett, hogy ezt a nehéz problémát megoldanánk, egyszerűen kerekítjük az $\mathbf{x}'$ minden koordinátaértékét a legközelebbi egész számra az (2). formulában, hogy megkapjuk az eredeti $\mathbf{x}$ vektor végső közelítését.

1.3. Linear Algebra and SQL in C++

1.3.1. SQLite

Az SQLite egy relációs adatbázis-kezelő rendszer, amely egyetlen C könyvtárban található. Sok adatbázis-kezelő rendszertől eltérően az SQLite adatbázis több platformon keresztül elérhető, és az alkalmazás egyetlen fájlként

[4]Például egy lineáris korlátozás amikor az $\mathbf{x}$ egy korlátozott tartományból származik. Ebben az esetben az optimalizálási probléma (OLS) továbbra is konvex másodfokú lineáris program marad, amelyet speciális megoldókkal könnyű megoldani.

[5]Ha $\mathbf{A}$ invertálható, akkor $\mathbf{A}^{-1} = \hat{\mathbf{A}}$. További részletekért lásd: https://en.wikipedia.org/wiki/Moore-Penrose_inverse.

[6]Ha $\mathbf{A}\mathbf{x} = \mathbf{b}$-nek több megoldása is létezik, akkor az OLS a legkisebb L_2 normáju értéket adja vissza.

[7]https://en.wikipedia.org/wiki/Ordinary_least_squares

lokálisan érheti el. Nincsenek önálló folyamatok, amelyekkel az alkalmazás kommunikálna, inkább az SQLite könyvtár (dinamikusan vagy statikusan) kapcsolódik az alkalmazáshoz, ami után egy SQLite adatbázist egyszerű függvényhívások segítségével lehet megnyitni, módosítani és lekérdezni.

A következő kódrészlet megnyitja az `adult.db` nevű helyi SQLite adatbázist, és kinyomtatja az egyes rekordok fizetési értékét. Callback függvénnyel lehet lekérdezni az SQL lekérdezés eredményét (MEGJEGYZÉS: Az `*out_data` a `callback` paraméterlistájában felhasználható adatok visszaadására a hívó függvénynek).

```
#include <iostream>
#include <sqlite3.h>

using namespace std;

// Call-back function for the execution of SQL queries
// (out_data can be used to return data to the caller function)
static int callback(void *out_data, int argc, char **argv, char **
 azColName)
{
   for (int i = 0; i < argc; i++)
   cout << azColName[i] << " = " <<  (argv[i] ? argv[i] : "NULL") <<
 endl;

   return 0;
}

int main(int argc, char **argv)
{
   sqlite3 *db;
   char *zErrMsg = 0;

   // Open database (local file)
   if (sqlite3_open("adult.db", &db) != SQLITE_OK)
   {
      cerr << "Can't open database: " << sqlite3_errmsg(db) << endl;
      sqlite3_close(db);
      return 1;
   }
   // Execute SQL statement
   if (sqlite3_exec(db, "SELECT salary FROM adult;", callback, 0, &
 zErrMsg) != SQLITE_OK)
   {
      cerr << "SQL error: " << zErrMsg << endl;
```

```
        sqlite3_free(zErrMsg);
    }
    sqlite3_close(db);
    return 0;
}
```

További részletek:
https://www.sqlitetutorial.net
https://www.sqlite.org/quickstart.html
https://www.sqlite.org/cintro.html

1.3.2. Armadillo

Az Armadillo egy lineáris algebra könyvtár a C++ nyelvhez. Hatékony mátrix alapszámítások végezhetők el vele, ugyanakkor egyszerű és könnyen használható felülettel rendelkezik. Lényegében felszereli a C++ nyelvet a `numpy`/`MATLAB` funkciók egy részhalmazával, és a mátrix műveleteket szinte ugyanolyan egyszerűvé teszi mint ezekben a keretrendszerekben.

A következő kódrészlet kiszámítja az $\begin{bmatrix} 1 & 2 & 1 \\ 2 & 3 & 0 \\ 1 & 3 & 2 \end{bmatrix}$ mátrix inverzét és transzponáltját, majd megszorozza a $(2, 5, 2)$ vektorral és az eredményt kiírja.

```
#include <iostream>
#include <armadillo>

int main(int argc, const char **argv)
{
    arma::vec b;
    b << 2.0 << 5.0 << 2.0;

    // arma::endr represents the end of a row in a matrix
    arma::mat A;
    A << 1.0 << 2.0 << 1.0 << arma::endr
    << 2.0 << 3.0 << 0 << arma::endr
    << 1.0 << 3.0 << 2 << arma::endr;

    A.print("A:");

    std::cout << "Inverse of A:\n";
    std::cout << arma::inv(A) << '\n';
    A.t().print("Transpose of A:");
```

```
    mat res = A * b;
    res.print("A*b=");
}
```

További részletek:
http://arma.sourceforge.net/docs.html
https://fossies.org/linux/armadillo/examples/example1.cpp

1.4. C++ fordítás Armadillo and SQLite könyvtárakkal

Az Armadillo és SQLite fejlesztői könyvtárakat először telepíteni kell[8]. Az Armadillo-t és az SQLite-t hívó `solver.cc` nevű C++ forrásfájl bármely Linux operációs rendszeren az alábbiak szerint fordítható és futtatható:

```
g++ -o solver solver.cc -larmadillo -lsqlite3
./solver
```

[8]Ubuntu alatt: `sudo apt install libsqlite3-dev libarmadillo-dev`

2. Feladatok

A feladat egy lekérdezés-auditor implementációja C++ nyelven, és azon rekordok számának meghatározása (`adult.db` adatbázis alapján), amelyek privát attribútumának (fizetésének) értéke visszaállítható a `queries.sql`-ben specifikált lekérdezésekből. A `adult.db` egy SQLite adatbázis, amelyet a 1.3.1 szakaszban részletezett SQLite C/C++ felületen keresztül lehet megnyitni és lekérdezni. A csatolt `queries.sql` minden sora egyetlen lekérdezésből áll, annak eredményével együtt. Például,

```
SELECT SUM(salary) FROM adult WHERE education = "Some-college"; 1128053
```

a `queries.sql` állomány egy érvényes sora, ahol a lekérdezés eredménye 1128053 (azaz az adatbázis összes olyan személyének összesített fizetése, aki főiskolán tanult). A `adult.db` adatbázis egyetlen egész értékű privát attribútummal rendelkezik, amelynek neve: `salary`, és a `queries.sql`-ben található összes lekérdezést ezen az attribútumon kell kiszámítani (az összes többi attribútum nyilvános).

MEGJEGYZÉS: *Az összes lekérdezésre adott válaszokat az Armadillo-val érdemes kiszámítani, mivel a Moodle is Armadillo-val számolt eredményekhez hasonlítja majd a beküldött megoldást a kvíz során.*

Tipp: Az adatbázis gyors megismeréséhez használható a `sqlite` parancs is, amely egy egyszerű parancssori környezet az SQLite adatbázisok lekérdezésére és kezelésére: https://sqlite.org/cli.html

2.1. 1. feladat

Készüljön fel a lekérdezések auditálására, és hozza létre az $\mathbf{A}$ mátrixot (a `queries.sql` fájlból), $\mathbf{x}$ vektort (`adult.db` adatbázis alapján) és $\mathbf{b}$ vektort (`queries.sql`-ből)! Az $\mathbf{x}$ vektor csak a rekonstrukciós hiba kiszámításához lesz szükséges az utolsó feladatban.

1. Hozza létre a $\mathbf{b}$ vektort a lekérdezés eredményeiből. Mennyi a legkisebb lekérdezés-eredmény?

2. Hozza létre az eredeti $\mathbf{x}$ adatvektort. Mi a legkisebb/legnagyobb fizetés? Hány rekord van az adatbázisban?

3. Hozza létre az **A** mátrixot a 4000 lekérdezésből. Mennyi a lekérdezések által lefedett (kiválasztott) rekordok átlagos száma? Mennyi egy lekérdezés által miniálisan/maximálisan kiválasztott rekordok száma?

Tanácsok a megoldáshoz:

- Az egyes rekordok `adult.db`-ből történő beolvasásához meg kell adni egy lekérdezést, és végrehajtani az `sqlite3_exec` segítségével aminek egy callback függvényt kell megadni. A callback függvény végzi a lekérdezés eredményeinek tényleges feldolgozását, és az eredményt visszaadja az első argumentumán keresztül, amely egy pointer (a 1.3.1. fejezetben részletezett `*out_data`). Erre egy működőképes megközelítés egy tömb létrehozása (például egy STL-szerű `vektor` objektum), ennek átadása a callback függvénynek, és az ott történő feltöltése adatokkal. Fontos, hogy az átadott tömb írásához a callback függvényben pointer casting-ot kell végezni!
- A lekérdezés által kiválasztott rekordok azonosításához használható az `idx` attribútum az adatbázisban. Ez az egyes rekordokhoz rendelt sorszám. Például a következő lekérdezés visszaadja azon rekordok indexét, amelyeket a példa lekérdezés választott ki a 2. fejezet elején:

```
SELECT idx FROM adult WHERE education = "Some-college";
```

2.2. 2. feladat

A lineárisan független lekérdezések száma a visszaállítható rekordok (pontosabban az **x** koordinátaértékek) számának felső korlátja. Számítsa ki az **A** mátrix lineárisan független sorainak a számát!

Hány lineárisan független lekérdezés van a `queries.sql` első 1000/2000/3000/4000 lekérdezésből?

2.3. 3. feladat

Készen áll a lekérdezések auditálására! A gyakorlatban több különböző technika használható a $\mathbf{Ax} = \mathbf{b}$ megoldására, de most maradjon a 1.2. fejezetben leírt Ordinary Least Squares (OLS) módszernél, mert a Moodle is az ezzel számolt megoldásokkal hasonlítja össze a beküldött eredményeket a kvíz során (a különböző módszerek kissé eltérő eredményeket hozhatnak).

Auditálja az első 1000/2000/3000/4000 lekérdezést a `queries.sql` fájlban! Hány rekordot tud az OLS teljesen helyreállítani az egyes esetekben? Más szavakkal: hány $\mathbf{x}$ koordinátaértéket tud visszaállítani az OLS segítségével (vegye figyelembe, hogy az $\mathbf{A}$ és $\mathbf{b}$ méretei eltérnek az egyes esetekben)?

Tanácsok a megoldáshoz:

- Az Armadillóban található `solve` nem azonos az OLS módszerrel, mivel a teljesítmény optimalizálása érdekében a `solve` több módszer egyikét választja, ami nem feltétlen az OLS.
- Ne feledje, hogy az OLS kimenete valós értékű vektor, de a fizetési értékek egész számok a `adult.db` adatbázisban (azaz $\mathbf{x} \in \mathbb{Z}^n$). Ezért ne felejtse el kerekíteni az OLS kimenetét (lásd: 1.2. fejezetet)!
- Az Armadillo beépített függvénnyel rendelkezik a mátrix pszeudóinverzének kiszámítására, de a mátrix SVD dekompozíciójából is kiszámíthatja azt.

Hivatkozások

[1] Simson L. Garfinkel, John M. Abowd, and Christian Martindale. Understanding database reconstruction attacks on public data. *Commun. ACM*, 62(3):46–53, 2019.

[2] Shubha U. Nabar, Krishnaram Kenthapadi, Nina Mishra, and Rajeev Motwani. A survey of query auditing techniques for data privacy. In Charu C. Aggarwal and Philip S. Yu, editors, *Privacy-Preserving Data Mining - Models and Algorithms*, volume 34 of *Advances in Database Systems*, pages 415–431. Springer, 2008.