practica 1
guille authored 3 days ago

2fc31500

practica-1-es.ipynb 23.9 KB

<img
src="http://www.uoc.edu/portal/_resources/common/imatges/marca_UOC/UOC_Masterbrand.jpg", align="left">

M1.771 · Privacidad
Máster Universitario en Ciberseguridad y Privacidad
Estudios de Informática, Multimedia y Telecomunicación

Práctica 1

Para resolver esta práctica, tenéis que poner las soluciones en el mismo *notebook*, en las celdas de código que están en blanco habilitadas para ello. Después de cada celda de código en blanco hay una celda *markdown* en verde donde podéis poner la justificación de la respuesta dada en la solución. No es obligatorio, pero sí muy recomendable que justifiquéis las respuestas que dais, ya que eso facilita la corrección en caso de una respuesta incorrecta (podiendo valorar el planteamiento), y puede ayudar a descartar posibles sospechas sobre copias.

Publicación de microdatos

Trabajamos en una biblioteca municipal y nos han encargado que supervisemos la publicación de datos sobre el préstamo de libros.

La biblioteca quiere hacer públicos datos sobre los libros que toman prestados sus usuarios con la idea de que investigadores puedan hacer estudios sobre el tipo de libros que se prestan en base a diversos factores como la edad o el lugar de residencia.

Para ello disponemos de unos datos sobre préstamo de libros en un fichero CSV con los siguientes atributos (columnas):

- `name` : nombre del usuario.
- `sex` : sexo del usuario, puede tomar el valor 'F' o 'M'.
- `age` : edad del usuario.
- `postal_code` : código postal de la residencia del usuario.
- `books` : lista de libros que el usuario ha tomado prestados.

Podemos leer el fichero CSV, que se encuentra en `data/books.csv` , como un `DataFrame` de *pandas* que podemos guardar en la variable `df_books` .

También crearemos un copia del `DataFrame` para tener a mano los valores originales en caso de que hagamos modificaciones en `df_books` .

Nota: funciones que pueden ser de utilidad:

- Para facilitar la lectura de la lista libros utilizamos la función `ast.literal_eval` que permite interpretar directamente el string como una estructura de Python (en este caso una lista).
- Con la función `DataFrame.info()` podemos ver las columnas del `DataFrame` que acabamos de leer y su tipo .

In [1]:
```
import ast
import pandas as pd

df_books = pd.read_csv("data/books.csv", dtype={'postal_code': str},
                       converters={'books': ast.literal_eval})
df_books.info()
df_books_original = df_books.copy()
```

Out [1]:
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20 entries, 0 to 19
Data columns (total 5 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   name         20 non-null     object
 1   sex          20 non-null     object
 2   age          20 non-null     int64 
 3   postal_code  20 non-null     object
 4   books        20 non-null     object
dtypes: int64(1), object(4)
memory usage: 928.0+ bytes
```

Ejerecicio 1 [5%]

El primer paso que nos plateamos es ver qué atributos no podemos publicar. Es decir, qué atributos consideraremos como **identificadores**. Determina qué atributos son identificadores y elimina esos atributos en `df_books`

Nota: algunas funciones que os pueden ser de utilidad:

- `pandas.DataFrame.drop` : elimina filas o columnas de un `DataFrame` .

Justificación:

Ahora nos planteamos si podemos publicar los datos que tenemos en `df_book` tal como están. Ya sabemos que esto puede ser peligroso. Además hemos descubierto que existe una pagina Web, *MyBooks*, que contiene opiniones sobre libros donde los usuarios puntúan los libros que han leído en una escala del 0 al 5. Esta Web publica dichos datos para permitir hacer estudios y desarrollar sistemas de recomendación.

Obtenemos una parte de los datos publicados por la web *MyBooks* en formato CSV con los siguientes atributos:

- `user_id` : identificador de usuario.
- `book` : título del libro.
- `rating` : puntuación asignada de 0 a 5.

MyBooks asegura que estos datos son anónimos ya que el identificador de usuario `user_id` es un número que sirve simplemente para poder agrupar los libros por usuario, y no revela información privada del mismo. Aunque ya se intuye que esto no tiene porque ser así, no vamos a tratar este problema concreto.

Podemos leer estos datos de *MyBooks* en un `DataFrame` de *pandas* en la variable `df_ratings` . Los datos están en `data/books_rating.csv` .

In [2]:
```
df_ratings = pd.read_csv("data/books_rating.csv")
df_ratings.info()
```

Out [2]:
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41 entries, 0 to 40
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   user_id  41 non-null     int64 
 1   book     41 non-null     object
 2   rating   41 non-null     int64 
dtypes: int64(2), object(1)
memory usage: 1.1+ KB
```

Las preferencias sobre las lecturas de cada usuario pueden considerarse como información privada. Conocerla puede revelar información relativa al usuario como sus gustos personales, ideología, orientación sexual, etc. Al no revelar más que un identificador por usuario podríamos considerar los datos de *MyBook* como relativamente seguros.

Ejercicio 2 [20%]

Supongamos que publicamos los datos de la biblioteca tal como los tenemos ahora. Si un atacante dispone de los datos de *MyBook* podría utilizar nuestros datos para intentar desanonimizar los datos de *MyBook*.

Es decir, utilizando los datos que tenemos ahora en `df_books` , podríamos obtener información sobre los usuarios contenidos en `df_ratings` . Si es así querría decir que podemos asociar a cada identificador del conjunto de datos `df_ratings` los atributos de `df_books` como *postal_code*, *age* o *sex*.

En este ejercicio se pide precisamente que determinéis qué registro de `df_books` corresponde a cada identificador utilizado en `df_ratings` . Para ello podemos comparar la lista de libros en `df_books` con la lista de los libros asociados a cada usuario (`user_id`) en `df_ratings` . La comparación la haremos por equivalencia del conjunto de libros. Es decir, los usuarios son el mismo si los dos conjuntos de libros contienen los mismo títulos.

Como resultado tenéis que dar para cada `user_id` de `df_ratings` la información que podáis de `df_books` .

Nota: una posible manera de solucionarlo sería creando dos diccionarios de Python a partir de los dos conjuntos de datos `df_books` y `df_ratings` . Estos diccionarios serían:

- `d_books` : diccionario tipo {index$_1$: list-of-books$_1$, index$_2$: list-of-books$_2$, ...} donde index$_i$ es el índice del registro i en `df_books` y list-of-books$_i$ la lista de libros de dicho registro.

- `d_ratings` : diccionario tipo {user_id$_1$: list-of-books$_1$, user_id$_2$: list-of-books$_2$, ...} donde user_id$_i$ es el identificador de un usuario en `df_ratings` y list-of-books$_i$ la lista de libros que tiene dicho usuario en `df_ratings` .

Una vez tenemos esto simplemente tenemos que comparar las listas de libros de ambos diccionarios para ver que índice de `df_books` le corresponde a que `user_id` de `df_ratings` . Podemos, incluso, hacer una función que realice todo este cálculo y retorne un diccionario de la forma: { index$_i$: user_id$_i$, ...}, donde index$_i$ es el índice en `df_books` del usuario i, y user_id$_i$ el `user_id` del mismo usuario en `df_ratings` .

- Comparación de conjuntos: las listas en Python mantienen el orden de los elementos, por lo que p.e. `[1,2,3] == [2,3,1]` retornará `False` . Si queremos compararlas sin tener en cuenta el orden podemos utilizar conjuntos (set): `set([1,2,3]) == set([2,3,1])` retornará `True` .

Justificación:

Ejercicio 3 [5%]

Con el resultado anterior, queremos ver si podemos reidentificar a usuarios de la biblioteca. Nos interesa obtener la lista de libros que han tomado prestados y sobre todo las valoraciones que da a cada libro. ¿Podemos reidentificar al siguiente usuario?:

- *Dave Smith*: sabemos que tiene 23 años.

Justificación:

Queremos ahora publicar los datos de la biblioteca que tenemos en `df_books` . Por una parte nos interesa que se pueda asociar información como edad, sexo o código postal, a los conjuntos de libros e incluso a las valoraciones de *MyBooks* pero no queremos que se pueda reidentificar a usuarios ni obtener información sobre ellos.

Si un atacante conoce la edad, sexo y código postal de un usuario concreto (por ejemplo un vecino) no debería poder llegar a conocer los libros que toma prestados en la biblioteca ni sus preferencias sobre los mismos.

Para ello aplicaremos varias técnicas de anonimización.

Ejercicio 4 [5%]

Aplicamos generalización al atributo `postal_code` para disminuir su precisión. Para ello lo vamos a generalizar a los 3 primeros dígitos. Es decir un código postal como "08193" pasará a ser "081**" en el atributo `postal_code` de `df_books` .

Justificación:

Ejercicio 5 [30%]

Finalmente aplicamos *rank swapping* al atributo `age` de forma independiente para protegerlo. Con esto conseguiremos que aunque un atacante conozca la edad de una persona de la base de datos no pueda saber a qué registro pertenece y por tanto enlazar con las preferencias de `df_ratings` .

Para poder comparar el resultado podéis mostrar los atributos *sex*, *age*, *postal_code*, de los `DataFrames` `df_books_original` y `df_books` .

Justificación:

Ejercicio 6 [5%]

Podemos ahora probar el mismo análisis o ataque descrito en el ejercicio 3 con el conjunto de datos protegidos `df_books` . ¿Podemos en este caso realizar la misma reidentifiación? Razona la respuesta.

Justificación:

Ruido aditivo

En este apartado vamos a estudiar la aplicación de ruido aditivo.

Utilizaremos un *dataset* muy popular en aprendizaje automático conocido como *Census Income Data Set* (o *Adult Dataset*). Podemos encontrar en el fichero `data/adult.data` el conjunto de datos en formato CSV. Este conjunto contiene datos sobre ingresos de población extraídos del censo de EUA del 1994. Cuenta con 32561 registros y 15 atributos.

Podemos leer el fichero en un `DataFrame` de *pandas* que guardaremos en la variable `df_census` y de todos los atributos que tiene nos quedaremos con los 5 atributos continuos: *age*, *fnlwgt*, *capital-gain*, *capital-loss*, *hours-per-week*.

In [3]:
```
df_census = pd.read_csv('data/adult.data',
                        names=[
                            'age', 'workclass', 'fnlwgt', 'education',
                            'education-num', 'marital-status', 'occupation',
                            'relationship', 'race', 'sex', 'capital-gain',
                            'capital-loss', 'hours-per-week', 'native-country',
                            'target'])

df_census.drop(['workclass', 'education', 'education-num', 'marital-status',
                'occupation', 'relationship', 'race', 'sex', 'native-country',
                'target'],
               axis=1, inplace=True)

df_census.info()
```

Out [3]:
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             32561 non-null  int64
 1   fnlwgt          32561 non-null  int64
 2   capital-gain    32561 non-null  int64
 3   capital-loss    32561 non-null  int64
 4   hours-per-week  32561 non-null  int64
dtypes: int64(5)
memory usage: 1.2 MB
```

Ejercicio 7 [20%]

Queremos aplicar ruido aditivo no correlacionado para proteger `df_census`. Aplicaremos ruido a todos los atributos. En este sentido consideramos todos los atributos como cuasi-identificadores.

Define una función llamada `additive_noise` que reciba como parámetro un `DataFrame` como `df_census`, el parámetro *p* que determina el nivel de ruido y retorne un `DataFrame` nuevo con el ruido añadido. El ruido aditivo, se obtiene de forma aleatoria de una distribución normal $N(\mu, \sigma^2)$ con una media $\mu = 0$ y una varianza $\sigma^2 = p\sigma_a^2$, donde σ_a^2 es la varianza de los valores originales.

El resultado debe ser del mismo tipo que los datos originales. Fijaros que todos los atributos son enteros por lo que el `DataFrame` resultante tendrá que tener también datos enteros.

Nota: algunas funciones que os pueden ser de utilidad:

- `numpy.random.normal` : permite obtener valores aleatorios de una distribución normal.
- `numpy.ndarray.round` : permite redondear los valores de un array de *numpy* (recordad que podemos operar sobre una columna de un `DataFrame` como si fuese un array de *numpy*).
- `numpy.ndarray.astype` : permite convertir el tipo de un array de *numpy*.

Justificación:

Ejercicio 8 [20%]

Para evaluar la pérdida de información que hemos introducido vamos a hacerlo en función del *Mean Squared Error (MSE)*. Podemos definir el cálculo de MSE entre dos `DataFrames` de *pandas* como:

In [4]:
```
def mse(df1: pd.DataFrame, df2: pd.DataFrame) -> float:
    return ((df1 - df2)**2).mean().mean()
```

Queremos proteger el conjunto de datos `df_census` utilizando el ruido aditivo del ejercicio 5, de manera que el MSE resultante de la protección sea menor que 3600000000 (36×10^8).

Importante: El nivel de ruido lo vamos a dar con un máximo de un decimal.

Para ver mejor el resultado también vamos a mostrar en un gráfico el MSE para niveles de ruido de 0.0 a 2.0 incrementando valores de 0.1. Es decir valores para $p = 0.0, 0.1, 0.2, 0.3, \ldots, 2.0$.

- ¿Que nivel de ruido máximo (parámetro *p*) podemos aplicar para obtener un MSE menor que 36×10^8?
- Muestra en un gráfica los valores de MSE para $p = 0.0, 0.1, 0.2, 0.3, \ldots, 2.0$.

Nota: para mostrar la gráfica podemos utilizar la función `plot` de `pandas`. Por ejemplo, si tenemos un diccionario de Python de la forma `m = { 0.0: 1234, 0.1: 12345, 0.2: 123456, ...}` podemos mostrar la gráfica utilizando una serie de pandas:

```
 ts = pd.Series(m.values(), index=m.keys())
ts.plot(xlabel='LabelX', ylabel='LabelY', grid=True, xticks=ts.index)
```

Justificación: