UNIVERSIDAD PONTIFICIA BOLIVARIANA
SEDE MEDELLÍN

DETECCIÓN DUPLICADOS

Samuel Pérez Hurtado
David Romero Rodríguez
Juan Alejandro Gómez Jaramillo

Facultad de Ingenierías
Universidad Pontificia Bolivariana, Sede Medellín
Ingeniería en Ciencia de Datos

El Data Set al que le realizamos el análisis tiene un total de 12000 filas, 8 columnas con datos sobre la información personal de una persona: nombre, dirección, correo electrónico, cumpleaños, ocupación, kilogramos y centímetros; también un id para identificar a cada una de ellas.

FRIL

Para realizar el análisis de duplicados en la herramienta Fril realizamos 3 experimentos con las siguientes condiciones:

- **EXPERIMENTO 1**: Agregamos los campos nombre, dirección, cumpleaños y ocupación para realizar el proceso; considerándolos relevantes y de ayuda para determinar los duplicados en el Data Set.
  Concluimos que el nombre y la dirección podrían llegar a ser los campos de mayor relevancia para la detección, por ello le asignamos mayores pesos. De igual manera utilizamos diferentes técnicas en cada uno de los campos, tratando de ajustar cada una de ellas a las características de estos:

  35% para el nombre – Técnica: Q grams distance
  30% para la dirección – Técnica: Street address distance
  17% para el cumpleaños – Técnica: Date distance con un rango de ± 2
  18% para la ocupación – Técnica: Q grams distance

- Para este caso usamos un umbral del 75%
- Para armar los bloques de candidatos utilizamos Soundex code y el atributo fue la dirección

RESULTADO: Obtuvimos en total 6924 registros duplicados, dado que varios de ellos se repetían 2 y hasta 3 veces. Se generaron duplicate score de 0 y entre 76 y 100.

Pudimos notar que la columna nombre era la que mayor problema de duplicados genera, dado que hay prefijos, abreviaturas y algunos errores ortográficos. Las otras 3 columnas en general tenían los datos casi que exactamente iguales.

Al final obtuvimos un Data Set con 5076 registros “limpios”; sin embargo, podemos indicar que dentro de esta cantidad seguramente habrán duplicados que el programa no detecto.

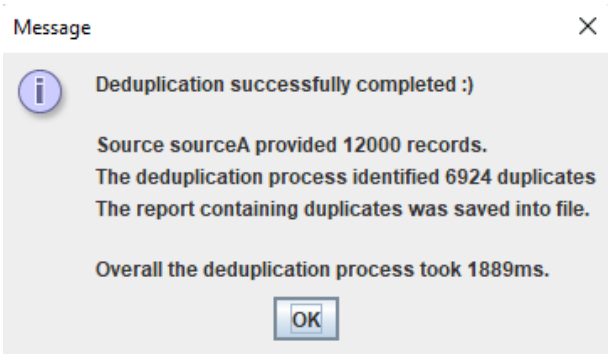

Imagen 1: Duplicados experimento 1

- **EXPERIMENTO 2**: Agregamos los campos id, nombre, dirección y ocupación para realizar el proceso; considerándolos relevantes y de ayuda para determinar los duplicados en el Data Set.

  Concluimos que el nombre sería el campo de mayor relevancia para la detección, por ello le asignamos el mayor peso. De igual manera utilizamos diferentes técnicas en cada uno de los campos, tratando de ajustar cada una de ellas a las características de estos:

0% para el id – Técnica: Equal fields Boolean distance
50% para el nombre – Técnica: Q grams distance
35% para la dirección – Técnica: Street address distance
15% para la ocupación – Técnica: Q grams distance

- Para este caso usamos un umbral del 75%
- Para armar los bloques de candidatos utilizamos Soundex code y el atributo fue la dirección

RESULTADO: Obtuvimos en total 6688 registros duplicados, nuevamente, varios de ellos se repetían 2 y hasta 3 veces. Se generaron duplicate score de 0 y entre 75 y 100.
En comparación con el experimento anterior, podemos indicar que la columna id efectivamente no es un buen campo para la detección de duplicados, pues genero una reducción en la cantidad de ellos. Los nombres igualmente seguían presentando el mayor problema.
Al final obtuvimos un Data Set con 5312 registros “limpios”; sin embargo, podemos indicar que dentro de esta cantidad seguramente habrán duplicados que el programa no detecto.

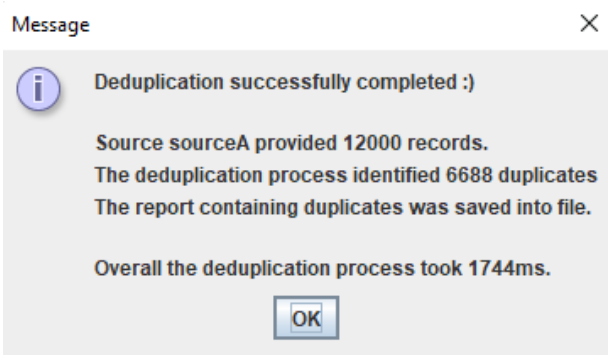

Imagen 2: Duplicados experimento 2

- **EXPERIMENTO 3**: Para este último caso quisimos aplicar una pequeña “estrategia” que nos permitiera que Fril pudiera comparar todos con todos. Cambiamos los valores de la columna número a un único valor para todas

las filas, luego lo exportamos como csv y al momento de armar los bloques de candidatos en Fril utilizamos Value of blocking attribute y el atributo fue el id.

0% para el id – Técnica: Q grams distance
40% para el nombre – Técnica: Q grams distance
35% para la dirección – Técnica: Street address distance
25% para la ocupación – Técnica: Q grams distance

- Para este caso usamos un umbral del 80%

RESULTADO: Como era de esperarse el software tuvo un tiempo de procesamiento muchísimo mayor que con los demás experimentos. Para nuestra sorpresa obtuvimos un resultado de 6664 duplicados, una cantidad menor con respecto al experimento 1 en donde no comparamos todos los valores con todos.
Como aspecto a resaltar, solo se generaron duplicate score de 0 y 100, es decir, se diferenció muy bien el duplicado del dato real.
Al final obtuvimos un Data Set con 5310 registros “limpios”; sin embargo, podemos indicar que dentro de esta cantidad seguramente habrán duplicados que el programa no detecto.

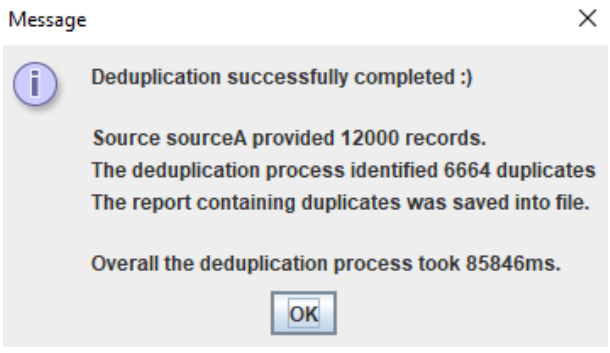

Imagen 3: Duplicados experimento 3

Teniendo en cuenta los experimentos que realizamos y los resultados obtenidos, podemos decir que, en general todos estos nos arrojaron resultados similares en cuanto al número de registros duplicados y las características de estos. Variables y configuraciones que realizamos en el software buscando la mejor forma de realizar el proceso es el gran diferenciador en los resultados. En ninguno de los experimentos utilizamos como campo los kilogramos ni los centímetros, pues consideramos que un mismo peso o altura la pueden tener muchas personas.

PYTHON

```
[3]
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12000 entries, 0 to 11999
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Id            12000 non-null  int64
 1   Nombre        12000 non-null  object
 2   Dirección     12000 non-null  object
 3   EmailAddress  12000 non-null  object
 4   Birthday      12000 non-null  object
 5   Occupation    12000 non-null  object
 6   Kilograms     12000 non-null  object
 7   Centimeters   12000 non-null  int64
dtypes: int64(2), object(6)
memory usage: 750.1+ KB
```

Imagen 4: Información general del Data Set

Al realizar la detección de los duplicados en Python obtuvimos un resultado bastante curioso. En el primer experimento, el resultado fue un Data Frame lleno de unos, como si todos los candidatos que nos ofreció el método indexer.full() realmente sí estuvieran duplicados. En ese primer experimento usamos técnicas como jarowinkler y damerau_levenshtein, dependiendo del tipo de dato de cada columna, en total obtuvimos 1185 duplicados.

```
# Iniciamos las comparaciones usando los métodos de recordlinkage.Compare

compare_cl = recordlinkage.Compare()
compare_cl.string('Nombre', 'Nombre', method='jarowinkler', threshold=0.85, label='Nombre')  #esta es una ven
compare_cl.exact('Birthday', 'Birthday', label='Birthday')
compare_cl.string('EmailAddress', 'EmailAddress', method='damerau_levenshtein', threshold=0.75, label='EmailA
compare_cl.numeric('Centimeters', 'Centimeters')
compare_cl.exact('Kilograms', 'Kilograms')
compare_cl.string('Dirección', 'Dirección', method='jarowinkler', threshold=0.75, label='Dirección')
compare_cl.string('Occupation', 'Occupation', method='jarowinkler', threshold=0.85, label='Occupation')
```

```
<Compare>
```

Imagen 5: Parámetros, umbrales y técnicas - experimento 1

```
features = compare_cl.compute(candidate_links, df)
features.head()
```

Visualize

	Nombre float64	Birthday int64	EmailAddress f…	3 float64	4 int64	Dirección floa…	Occupatic
(9000, 0)	1	1	1	1	1	1	
(9003, 3)	1	1	1	1	1	1	
(9006, 6)	1	1	1	1	1	1	
(9010, 10)	1	1	1	1	1	1	
(9012, 12)	1	1	1	1	1	1	

5 rows, showing 10 per page Page 1 of 1

Imagen 6: Data Frame – experimento 1

```
matches = features[features.sum(axis=1) > 5]

print(len(matches))
matches.head()
```

```
1185
```

Imagen 7: Cantidad duplicados – experimento 1

Al realizar el **segundo experimento** en Python, también obtuvimos un Data Frame lleno de unos, incluso al cambiar parámetros, umbrales y técnicas. En este experimento usamos q-grams, levenshtein y jaro. El resultado, al igual que en el primer experimento, fue de 1185 duplicados.

```
compare_cl = recordlinkage.Compare()
compare_cl.string('Nombre', 'Nombre', method='qgram', threshold=0.75, label='Nombre') #esta es una ventaja d
compare_cl.exact('Birthday', 'Birthday', label='Birthday')
compare_cl.string('EmailAddress', 'EmailAddress', method='levenshtein', threshold=0.85, label='EmailAddress')
compare_cl.numeric('Centimeters', 'Centimeters')
compare_cl.exact('Kilograms', 'Kilograms')
compare_cl.string('Dirección', 'Dirección', method='jaro', threshold=0.80, label='Dirección')
compare_cl.string('Occupation', 'Occupation', method='jarowinkler', threshold=0.75, label='Occupation')
```

```
<Compare>
```

Imagen 8: Parámetros, umbrales y técnicas - experimento 2

```
features = compare_cl.compute(candidate_links, df)
features.head()
```

Visualize

	Nombre float64	Birthday int64	EmailAddress f…	3 float64	4 int64	Dirección floa…	Occupatic
(9000, 0)	1	1	1	1	1	1	
(9003, 3)	1	1	1	1	1	1	
(9006, 6)	1	1	1	1	1	1	
(9010, 10)	1	1	1	1	1	1	
(9012, 12)	1	1	1	1	1	1	

5 rows, showing 10 per page Page 1 of 1

Imagen 9: Data Frame – experimento 2

```
features.sum(axis=1).value_counts().sort_index(ascending=False)
```

```
7.0    1180
6.0       5
dtype: int64
```

Podemos ver que cambiando algunos parámetros, obtenemos los mismos resultados

Imagen 10: Cantidad duplicados – experimento 2

Una ventaja de Python sobre Fril es que, en Python, podemos asignar un umbral por campo o por variable, mientras que en Fril debemos asignar el umbral para todos los registros de la fila. Por otro lado, la amplia comunidad y desarrollo de Python permite que las nuevas técnicas que se vayan desarrollando sean

implementadas fácilmente, mientras que en Fril esto ya no es posible dado que es un software que no volvió a tener actualizaciones. Sin embargo, Fril resulta una herramienta muy práctica al momento de enfrentarnos a una situación real en donde necesitemos buscar duplicados a un conjunto de datos.