Análisis Comparativo Entre Herramientas de Data Science con Pandas y SQL

A. Vekselman, *Instituto Tecnológico de Buenos Aires*

***Abstract*—En este trabajo se propone realizar un análisis comparativo entre la librería Pandas de Python y la herramienta SQL para análisis de datos. Se intentarán responder preguntas sobre en qué casos usar uno u otro y cuál conviene a la hora de analizar distintos sets de datos. Para ello, se utilizarán métricas como la velocidad de ejecución, las restricciones de memoria y las herramientas proporcionadas para la minería de datos. Ambos lenguajes son ampliamente utilizados en el mundo del análisis de datos, por lo cual se espera que cada uno tenga sus ventajas y desventajas respectivas. Luego de cada aspecto, se hará una conclusión parcial de los resultados obtenidos de la comparación. Finalmente, se expresará una conclusión general a modo de cierre, en la cual se intentará responder a las preguntas planteadas previamente.**

I. Introducción

Para empezar a realizar un análisis comparativo entre la librería Pandas de Python y SQL, es necesario tener en claro algunos conceptos previos relacionados con su uso y funcionamiento.

En primer lugar, ¿qué es 'análisis de datos', o *data analysis*, o *data science*? Según la universidad de Illinois,

Análisis de datos es el proceso de sistemáticamente aplicar técnicas estadísticas y/o lógicas para describir e ilustrar, condensar y resumir, y evaluar datos. [1]

¿Qué quiere decir esto? Esta expresión hace referencia al manejo de datos como el tratamiento de un conjunto de información con un hilo conector, al cual se le pueden aplicar diversas técnicas para transformar dicha información y convertirla en datos. Éstos, a su vez, contienen información útil (o no) sobre el tema de la investigación actual. Como mencionan Shamoo y Resnik, escritores del libro "Conducta responsable para la investigación",

Los procedimientos analíticos variados proveen una forma de esquematizar inferencias inductivas desde los datos y distinguir señal (el fenómeno de interés) del ruido (fluctuaciones estadísticas) presente en los datos.[2]

Esto quiere decir que el análisis de datos no es un fin en sí, sino que se utiliza como herramienta en las investigaciones para poder llevarlas a cabo efectivamente, estableciendo los patrones y relaciones que, sin herramientas como Pandas o SQL, serían muy difíciles de establecer. Así, con las tecnologías mencionadas, se logran obtener datos más valiosos, precisos y organizado a partir del grueso de la información existente.

II. Introducción a Pandas

Pandas es una librería de Python basada en estructuras llamadas *dataframes*, las cuales podrían pensarse como tablas de información donde se almacenan los datos de forma organizada. Sus columnas tienen un encabezado para distinguirlas entre sí, y sus filas tienen índices.

En base a este concepto es que se construye el funcionamiento de esta librería. Un ejemplo:

```
import pandas as pd

df = pd.DataFrame()

df["First"] = range(4,7)
df["Second"] = [2*i for i in range(4,7)]

print (df)

"""
Imprime:

   First  Second
0      4       8
1      5      10
2      6      12

"""
```

Fig. 1. Primeros pasos con Pandas.

En la figura 1 se observan los primeros pasos para utilizar la librería Pandas. Primero se la debe importar, utilizando convencionalmente el prefijo "pd". Para crear un dataframe, simplemente se debe llamar a la clase *DataFrame()*.

En este caso, la variable 'df' hace referencia a un dataframe, y se le crearon dos columnas. Se puede ver en la figura el dataframe impreso. Como se mencionó antes, la estructura es

[1] Facultad de desarrollo y centro instruccional de diseño, *Conductas responsables para la*, https://ori.hhs.gov/education/products/n_illinois_u/datamanagement/datopic.html, Universidad Norteña de Illinois.

[2] Shamoo, Resnik, *Conductas Responsables para la Investigación*, tercera edición, 2003.

similar a la de una tabla, con títulos en las columnas e índices en las filas. Internamente, el funcionamiento del dataframe es como el de un diccionario de Python, por lo cual para crear nuevas columnas basta con asignarle una serie de datos (valor) al título de la columna (clave).

Se puede observar claramente la facilidad de escritura que provee Pandas, abstrayendo al programador de los procesos que se efectúan por detrás.

Para manipulación de los datos, como seleccionar una sola columna, se puede hacer lo siguiente:

```
df["First"] = range(4,7)

print (df["First"])
print (df.First)

"""
Ambas imprimen:

0    4
1    5
2    6
Name: First, dtype: int32

"""
```

Fig. 2. Selección de columnas en Pandas.

Se puede observar que, para seleccionar una columna, basta con hacer referencia al valor guardado en esa columna (equivalentemente a un diccionario) o tratar al nombre de la columna como un dato miembro de la variable 'df'.

Para hacer filtrado sobre las columnas, se puede efectuar lo siguiente:

```
import pandas as pd
df = pd.DataFrame()
df["First"] = range(4,8)

print (df[df["First"] > 5].reset_index(
drop = True))

"""
Imprime:

   First
0      6
1      7
"""
```

Fig. 3. Filtrado por condición lógica en Pandas.

Como se observa, la sintaxis para filtrar datos a partir de una condición lógica es casi intuitiva, y resulta de aplicar el operador [ ] al dataframe con la condición lógica.

Nota: la función 'reset_index' se utiliza para, justamente, arreglar los índices de forma tal que coincidan con los valores naturales de un dataframe.

Otro concepto interesante de Pandas, que también se encuentra presente en SQL, es la unión de columnas a partir de cierta condición o dato en común. Ejemplo:

```
First = list(range(4,8)) * 2

df["First"] = First
df["Second"] = df["First"].apply(lambda
x: 2*x)

df2 = df.groupby("First")["Second"].sum
().reset_index()
print (df2)

"""
Imprime:
   First  Second
0      4      16
1      5      20
2      6      24
3      7      28
"""
```

Fig. 4. Groupby y apply en Pandas.

En la figura 4 se pueden ver dos de las funciones más importantes que presenta la librería: apply y groupby.

La primera es un método de las columnas del dataframe, cuyo funcionamiento consiste en aplicar cierta función a cada una de las filas de dicha columna. En este caso, se definió la columna "Second" a partir de efectuar un "apply" sobre la columna "First" con la función para multiplicar por dos.

La segunda de las funciones, groupby, recibe una o varias columnas y, como su nombre lo indica, agrupa las distintas columnas a partir de los valores coincidentes. En este caso, estaban repetidos los valores 4, 5, 6 y 7, por lo que la lista resultante tiene sólo cuatro entradas, correspondientes a cada valor distinto en la columna sobre la cual se agrupó. Luego, se aplicó la función 'sum'. Este formato es muy utilizado durante la extracción de datos de un dataframe.

El procedimiento sería: agrupar por una o varias columnas, luego elegir la o las columnas sobre las que se desea operar para extraer datos, y finalmente aplicar la función que se desea. Como en este caso se aplicó la función 'sum', que suma todos los valores, el resultado final de la columna 'Second' consiste en la suma de todos los valores de la columna 'Second' original que tuvieran el mismo valor en su correspondiente celda de la columna 'First'. Así, como había dos filas en 'First' con valor 4, y ambas tenían el valor 8 en 'Second', entonces el valor final de 'Second' en la fila que tenga 'First' igual a 4 será 8 + 8 = 16.

Sólo para ilustrar las facilidades que brinda Pandas, la figura 5 corresponde al código necesario para efectuar el procedimiento anterior si se estuviera trabajando con listas y diccionarios nativos de Python:

```
# Defino las variables a utilizar
df2 = {}
temp = {}
```

```
# Loopeo por cada uno de los valores en
'First'
for i in range (len(df['First'])):

 # Si es uno nuevo, lo cargo en 'temp'.
    if not temp.get(df['First'][i]):
        temp[df['First'][i]] = df['Seco
nd'][i]

   # Si ya había aparecido, lo sumo al
valor de 'temp'.
    else:
        temp[df['First'][i]] += df['Sec
ond'][i]

# Las keys del diccionario serán los
valores de 'First'.
df2['First'] = list(temp.keys())

# Los values del diccionario serán los
valores de 'Second'.
df2['Second'] = list(temp.values())
```

Fig. 5. Ejemplo de programa con misma funcionalidad que brinda Pandas.

Como se ve en la figura 5, el procedimiento que antes llevó una sola línea resulta algo más complejo cuando se lo intenta imitar con funcionamiento nativo. Como se verá más adelante, la librería Pandas está optimizada para realizar cada paso de forma mucho más rápida y eficiente, siempre y cuando la cantidad de datos se mantenga dentro de ciertos límites.

III. Análisis

A. *Velocidad (complejidad)*

Pandas

Para ver los resultados en términos del tiempo que tarda Pandas, se decidió tomar como parámetro la complejidad, es decir, cómo varía el tiempo que se tarda en ejecutar el algoritmo a medida que se incrementa el número de datos. El código a evaluar fue el siguiente:

```
df = pd.DataFrame({"First" : np.array(range(
1,value)) % 10 + 1})
df["Second"] = df.First.apply(lambda x:
np.log(x))

df2 = df.groupby(["First"])["Second"].
mean().reset_index()
```

Fig. 6. Código utilizado para evaluar velocidad de Pandas.

Donde el prefijo ‘pd’ hace referencia a Pandas y el prefijo ‘np’ hace referencia a la librería NumPy.

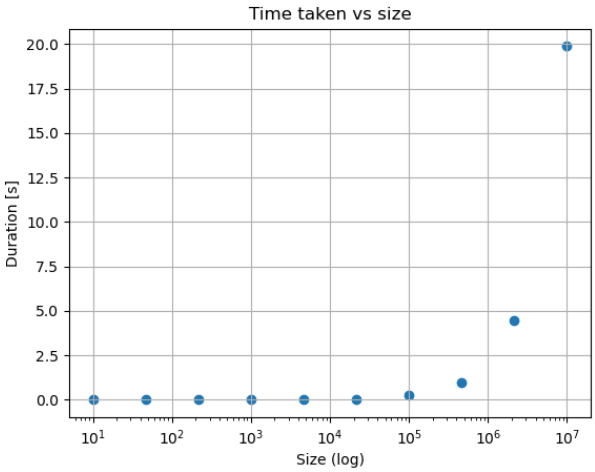

Fig. 7. Tiempo de ejecución en segundos vs cantidad de datos en Pandas.

El procedimiento que se realizó fue ir variando la cantidad de datos e ir midiendo el tiempo que el programa tardaba en ejecutar el código.

Los resultados obtenidos se pueden ver en la figura 7.

Al hacer los análisis, se observa una clara tendencia lineal (en este caso se mantuvieron las escalas logarítmicas para que

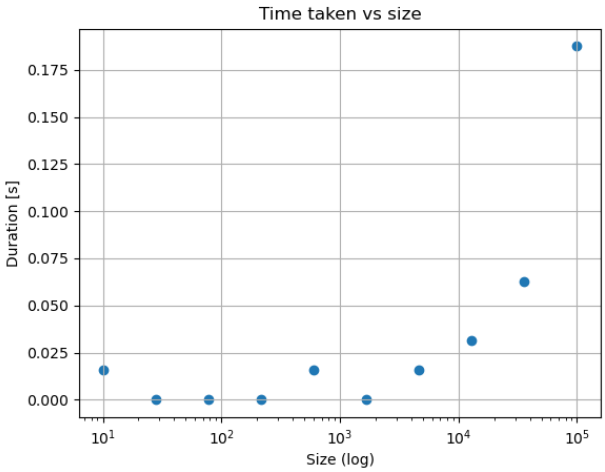

Fig. 8. Tiempo de ejecución en segundos vs cantidad de datos en Pandas.

las escalas concuerden con las de SQL). Poniendo atención a la zona de tamaños entre 10 y 10^5, como en la figura 8, se puede ver que la tendencia sigue manteniendo esta misma forma.

Esto lleva a pensar que el algoritmo utilizado por Pandas tiene una complejidad del orden de O(n), lo cual no es una característica ideal para una herramienta que usualmente trabaja con grandes cantidades de datos (n grande).

Nota: Para los análisis de duración se utilizó la librería de Python *Time*.[3]

[3] Python.org, *The Python Standard Library, Generic Operating System Services*, https://docs.python.org/3/library/time.html, sin autor especificado.

SQL

Equivalentemente, el mismo programa fue escrito en SQL. Para medir su velocidad, se varió la cantidad de datos con las que se trabajaba, y los resultados obtenidos fueron los siguientes:

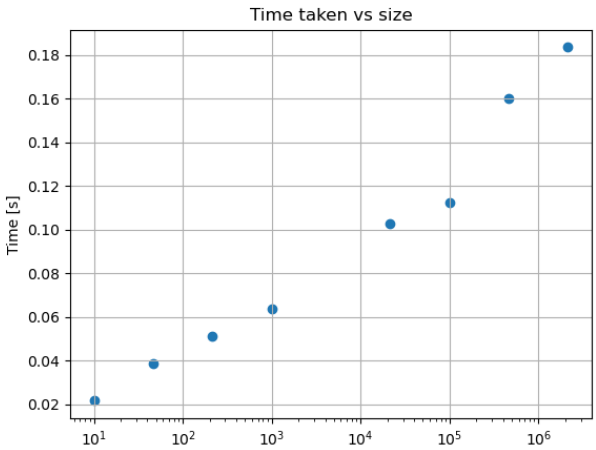

Fig. 9. Tiempo de ejecución en segundos vs cantidad de datos en SQL.

Se ve claramente en la figura 9 cómo los resultados obtenidos con SQL son de la forma O(log(n)), ya que la forma de la curva es una recta en escala logarítmica.

Nota: Para los análisis de duración se utilizó la herramienta nativa de SQL de manejo de día y hora. [4]

COMPARACIÓN

Como se vio en las secciones anteriores, los resultados obtenidos con Pandas y SQL en cuanto a complejidad fueron muy dispares.

Por un lado, para cantidades pequeñas de datos ambas herramientas tuvieron un desempeño similar. Incluso Pandas fue superior a SQL para algunos valores de tamaño.

Sin embargo, a medida que la cantidad de datos fue aumentando considerablemente (escala logarítmica), el desempeño de Pandas fue decayendo de manera lineal, mientras que el de SQL lo hizo de manera logarítmica, como se explicó. Para valores cercanos al millón de datos, o incluso más, SQL realizó el procesamiento en un tiempo considerablemente menor a un segundo, mientras que Pandas tardaba alrededor de 20. Esto da la pauta de que, si se quiere trabajar con grandes cantidades de datos, la opción para elegir debe ser SQL.

Los resultados obtenidos durante esta comparación son consistentes con las evidencias que se pueden encontrar en búsquedas en Internet, donde el uso que se le suele dar a estas herramientas varía mucho dependiendo del tamaño del set de datos. Por lo general, para grandes sets se prefiere la eficiencia de SQL, pero para sets pequeños se prefiere Pandas por su versatilidad al usarla en conjunto con otras herramientas de Python.

B. Memoria

PANDAS

Para evaluar el uso de la memoria, se decidió utilizar el mismo procedimiento del punto anterior, variando la cantidad de datos a utilizar. Los resultados que se obtuvieron fueron los siguientes:

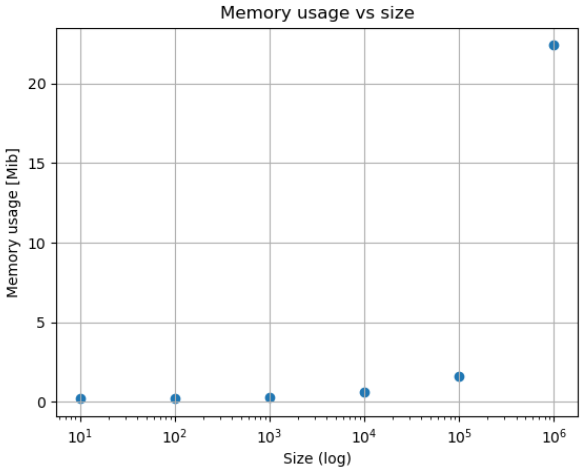

Fig. 10. Mib de memoria utilizados vs cantidad de datos en Pandas.

Similar al caso de la duración, se observa una tendencia creciente importante. Al realizar el análisis en escala lineal, se observó una tendencia entre lineal (O(n)) y exponencial, lo cual es un desempeño muy poco favorable para Pandas.

Además, al intentar subir todavía más la cantidad de datos a utilizar, ocurrió un *Memory Error*, indicando que la cantidad de memoria que Python tiene permitido usar era menor a la cantidad necesaria para ejecutar el programa.

Nota: Para realizar los análisis de uso de memoria se utilizó la librería *memory_profiler*.[5]

SQL

Análogamente, se utilizó el mismo código que antes para realizar los análisis. En la figura 11 se observan los resultados obtenidos.

[4] spencer7593, The Moot, StakOverflow, https://stackoverflow.com/questions/11675077/measure-the-time-it-takes-to-execute-a-t-sql-query/11675263, 26/07/2012.

[5] F. Pedregosa, P. Gervais, "Memory Profiler", PyPI.org, https://pypi.org/project/memory-profiler/.

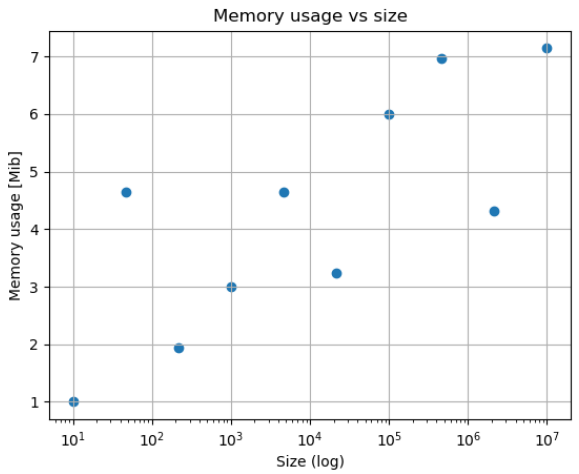

Fig. 11. Mib de memoria utilizados vs cantidad de datos en SQL.

Nuevamente, dado que en la escala logarítmica se ve una distribución con una tendencia claramente lineal, se puede establecer que el crecimiento del uso de memoria con respecto a la cantidad de datos tiene forma logarítmica.

Nota: Para hacer los análisis de uso de memoria se utilizó la interfaz de manejo de memoria de la IDE SSMS de Windows.[6]

Comparación

Como se observó, los resultados del análisis de uso de memoria fueron tan concluyentes como los de velocidad. El desempeño de SQL fue claramente superior al de Pandas para cantidades grandes de datos, mientras que Pandas tuvo la ventaja con cantidades pequeñas.

De esta manera, con los resultados anteriores y los actuales, se puede ir formando una idea de cómo se suelen complementar ambas herramientas en análisis de datos. Como se mencionó, es muy común hacer el procesado del grueso de los datos con SQL, filtrarlos y luego trabajar con Pandas para hacer los desarrollos más específicos con sets de datos más reducidos. Así se logra un balance ideal, aprovechando las características más positivas de cada lenguaje.

Cabe destacar que Pandas se suele utilizar muy en conjunto con las librerías NumPy y MatplotLib, para procesado y visualización de los datos, respectivamente. Estas son dos herramientas muy poderosas que provee Python, y las cuales, si bien no se tocarán en esta monografía, son ventajas importantes que posee Python.

C. *Minería de datos*

El libro "Introducción a la Minería de Datos", la minería de datos se puede definir de la siguiente manera:

La minería de datos es una parte integral del descubrimiento de conocimiento en bases de datos, lo cual es el proceso de convertir datos crudos en información útil. (...) Este proceso consiste en una serie de pasos transformativos, desde preprocesamiento hasta post procesamiento de resultados del minado de datos.[7]

¿Qué quiere decir esto? Básicamente, que el minado de datos es el primer paso antes de poder trabajar satisfactoriamente con cualquier set de datos lo suficientemente grande. La minería de datos se basa en, como su nombre lo indica, realizar una especie de excavación entre los datos para encontrar los diamantes, es decir, la información relevante y los patrones que sigue dicha información.

Para hacer esto, tanto Pandas como SQL poseen herramientas que resultan clave a la hora de trabajar con bloques de información en primera instancia desconocida.

Pandas

El enfoque que hace Pandas sobre la minería de datos se basa en ir analizando los dataframes desde distintos puntos de vista, con distintos parámetros, para construir una imagen más o menos representativa de la información. Aquí se procederán a explicar las funciones que se utilizan.

El proceso de minado de datos en Pandas comienza con la función más sencilla, 'head'. Como se observa en la figura 12, esta función simplemente devuelve las primeras filas del dataframe. Por defecto el valor es 5, pero éste se puede configurar al tamaño que se quiera, pasándole la cantidad como parámetro a la función. De esta forma, ya se comienza a tener una idea del formato y el tipo de información que se maneja.

```
df = pd.DataFrame()
df["First"] = np.arange(9)
df["Second"] = df["First"].apply(np.sin
)
print(df.head())

"""
Imprime:
   First    Second
0      0  0.000000
1      1  0.841471
2      2  0.909297
3      3  0.141120
4      4 -0.756802
"""
```

Fig. 12. Uso de función 'head' en Pandas.

Luego se procede a utilizar funciones, como se ve en la figura 13, como 'isna' o 'isnull' para tener una idea de con cuántos datos útiles se está trabajando, qué tan completa está la tabla y qué tanta información falta. Si bien en primera instancia esto podría parecer trivial, al final de este apartado se hará un comentario al respecto para mostrar por qué no lo es.

[6] Microsoft Documentation, SQL Documentacion, "Monitor and Troubleshoot Memory Usage", https://docs.microsoft.com/en-us/sql/relational-databases/in-memory-oltp/monitor-and-troubleshoot-memory-usage?view=sql-server-ver15#bkmk_Monitoring, 03/17/2017.

[7] M. Steinbach, P. Tan, V. Kumar, "Introduction to Data Mining", Addison-Wisley, 2005.

```
df = pd.DataFrame({
     "First" : [1,2,3,pd.NA],
     "Second" : [4,5,6,pd.NA]
})
print(df.isna())

"""
Imprime:
   First  Second
0  False   False
1  False   False
2  False   False
3   True    True
"""
```

Fig. 13. Uso de función 'isna' en Pandas.

Una vez entendida la topología de los datos, se procede a realizar un análisis más en profundidad. Se puede continuar con el dato miembro 'dtypes', para conocer los tipos de datos, y la función 'describe'. Esta última devuelve una tabla, como se ve en la figura 14, con información de primer orden que puede o no resultar relevante para el análisis, pero de la cual definitivamente se podrá extraer información útil. Tal como dice la figura 14, este comando provee información como la media de cada columna, el desvío, la cantidad de datos, el máximo, mínimo y los cuartiles.

```
df = pd.DataFrame({
     "First" : range(5),
     "Second" : range(7,12)
})

print(df.describe())

"""
Imprime:
           First     Second
count   5.000000   5.000000
mean    2.000000   9.000000
std     1.581139   1.581139
min     0.000000   7.000000
25%     1.000000   8.000000
50%     2.000000   9.000000
75%     3.000000  10.000000
max     4.000000  11.000000
"""
```

Fig. 14. Uso de función 'describe' en Pandas.

Dado que una ventaja muy importante de Pandas es la combinación con otras herramientas de Python, se procederá a mencionar algunas de ellas, y cómo utilizarlas en el proceso de minado de datos.

Una de las herramientas que suelen utilizarse es la ya mencionada MatplotLib, para graficar los datos y obtener una clara visualización de las distribuciones y relaciones. Con comandos de una línea se pueden hacer gráficos de barras, de torta, histogramas y obviamente ploteos convencionales. Los gráficos hechos en secciones anteriores de este mismo trabajo, de hecho, fueron realizados con esta librería.

En combinación con la información que se pueda extraer de MatplotLib, se utilizan librerías como TensorFlow, ScikitLearn o StatsModels, basadas en el uso de herramientas estadísticas y de *machine learning* (ML) o aprendizaje automático, tanto para la descripción de los sets de datos como también para la predicción de futuros datos a partir de los existentes.

Cabe destacar que todas las funciones y herramientas mostradas en esta sección pueden ser combinadas de varias formas distintas, e incluso pueden recibir distintos parámetros, para ver sólo ciertos aspectos de ciertas partes de la información, en caso de considerarse necesario. El enfoque visto durante el ejemplo fue de carácter ilustrativo, y más bien sencillo en naturaleza.

A modo de comentario final de esta sección, se considera importante remarcar las diferencias que pueden tener entre sí sets con distintas topologías cuando se los intenta fittear en un modelo con las librerías mencionadas previamente. A mayor cantidad de entradas vacías, el grado de error que presentarán estos modelos crecerá de manera muy acelerada. Y dado que tanto las herramientas de descripción estadística como los modelos de predicción a partir de ML son las bases que definen a Python como uno de los lenguajes más populares para el análisis de datos, es imperativo que la topología de los sets de datos sea la adecuada para que dichas herramientas funcionen de manera óptima.

SQL

Para minería de datos, la IDE de Microsoft para SQL provee un *wizard* de minería de datos, enfocado principalmente a la abstracción del programador de los algoritmos. Así, la herramienta de SQL de Microsoft viene equipada con distintas implementaciones de los algoritmos usualmente utilizados al momento de analizar y predecir el comportamiento de distintos sets de datos. Algunos de los algoritmos y estructuras más importantes, usados no sólo en SQL, sino en todas las plataformas, son la regresión lineal, los árboles de decisión, la agrupación en clusters, y hasta otros más complejos como las redes neuronales, como se observa en la figura 15.

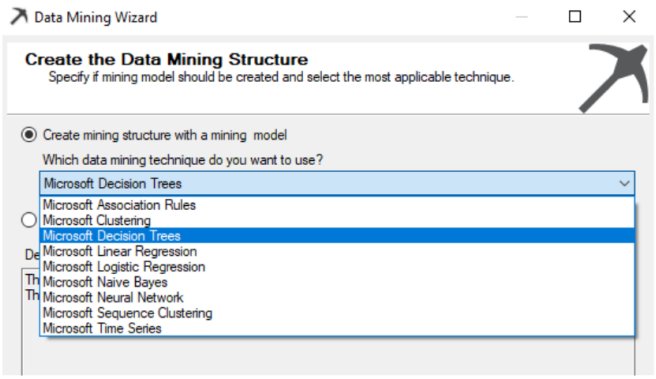

Fig. 15. Data Mining Wizard en Microsoft SQL.

Estos algoritmos mencionados previamente tienen como finalidad tanto la clasificación como la predicción del

comportamiento de los modelos. Muchas de las implementaciones de Microsoft de estos algoritmos son ampliamente utilizadas en el ambiente de análisis de datos, ya que brindan metodologías eficientes para la resolución del problema de minería de datos. Cabe destacar que, a su vez, estos algoritmos no son excluyentes entre sí, sino todo lo contrario; se utilizan en conjunto para lograr una descripción detallada de la información y su comportamiento a través de enfoques radicalmente (o parcialmente, dependiendo del caso) distintos.

Por cuestiones de extensión, dado que tanto los algoritmos a utilizar requieren explicaciones estadísticas que exceden al alcance del trabajo, se procederá directamente a mostrar cómo se vería el resultado de aplicar uno de estos algoritmos a dos series de datos ya preestablecidas.

A su vez, por simplicidad, el algoritmo elegido fue la regresión lineal, en particular una regresión lineal monovariable. Si bien es el más sencillo, y muchas veces no hace los ajustes necesarios para la descripción del problema, en minado de datos con relación lineal sí hacen un buen análisis y, por consiguiente, una buena predicción.

Nota: Dado que Python provee herramientas más sencillas para el ploteado de datos, la información obtenida de SQL se ha exportado y graficado en Python.

En un primer caso, si se observa la figura 16 se puede observar cómo la regresión lineal no ajusta bien a los datos reales, presentando fluctuaciones importantes entre los valores predichos por el modelo y los datos del dataset.

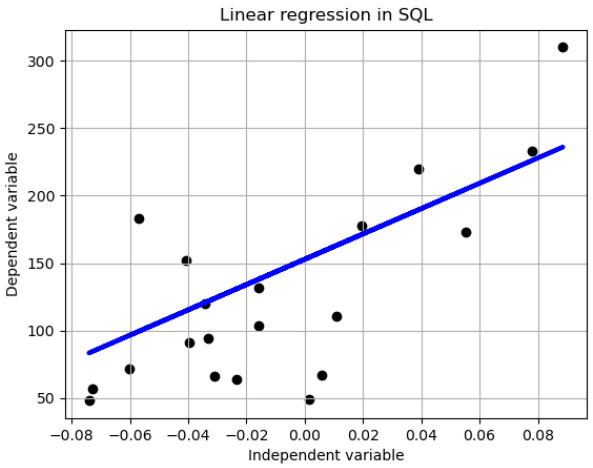

Fig. 16. Regresión lineal con SQL con R^2 cercano a 0.

El caso observado en la figura 16 es el más común, ya que gran cantidad de los problemas que se presentan en data science son del tipo no líneal, o al menos no monovariable. De todas maneras, se ve cómo el algoritmo de SQL intenta encontrar relaciones entre los distintos datos y produce la línea azul que se ve en el gráfico. Este algoritmo de regresión lineal es también soportado por varias librerías de Python, y también por otras herramientas de Microsoft como Microsoft Excel.

Por otro lado, hay situaciones en las que los datos sí cumplen una relación más estrechamente lineal, por lo cual el modelo de regresión lineal de SQL para minería de datos es ideal para su modelización. Como se ve en la figura 17, la curva se ajusta casi perfectamente, más allá de las pequeñas fluctuaciones.

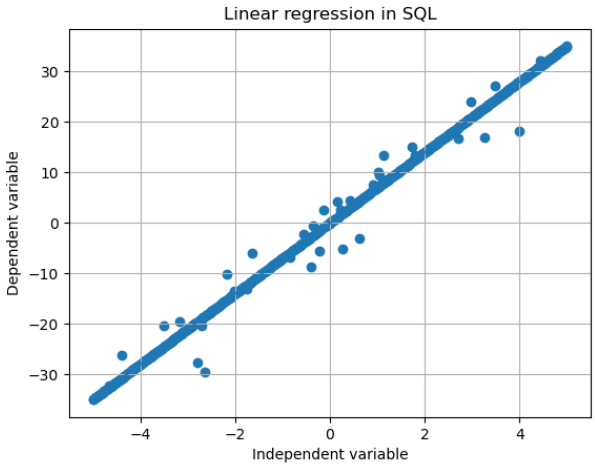

Fig. 17. Regresión lineal con SQL con R^2 cercano a 1.

De esta manera, el algoritmo a utilizar depende mucho del problema, y la minería de datos se basa justamente en intentar encontrar qué modelo representará mejor a los datos y qué relaciones se pueden encontrar entre ellos.

La herramienta de minería de datos de SQL es muy poderosa a la hora de encontrar estas relaciones, y a la vez tiene la ventaja de su simplicidad de uso, ofreciendo una GUI integrada en la IDE con la finalidad de abstraer de su funcionamiento interno y mostrar sólo la información importante.

Comparación

Como se vio en las respectivas secciones, ambas herramientas, Pandas y SQL, proveen metodologías para la minería de datos. Sin embargo, en este caso la ventaja se la lleva Python, simplemente por el hecho de tener una versatilidad mucho mayor. Así como se explicó y se mostró el ejemplo del modelo de regresión en SQL, como también se nombraron algunos otros modelos, Python brinda esta funcionalidad en librerías como TensorFlow o ScikitLearn, previamente mencionadas. E incluso Python lo lleva un paso más allá, con modelos que en SQL no aparecen. Ejemplos de esto son el algoritmo de “K” vecinos más cercanos o las máquinas de vectores de soporte. Además, Pandas posee la ventaja de tener acceso a toda la estructura de Python, que brinda un entorno de incontables herramientas enfocadas en la minería y análisis de datos.

De esta manera, la versatilidad se vuelve un enfoque fundamental a la hora de analizar, ya que hay tantos tipos de análisis como datasets distintos. Y dado que lo ideal es encontrar el modelo que mejor se adapte al caso particular que se tiene, una herramienta con mayor cantidad de tecnologías será la que logre encontrar el balance de modelos que mejor describa los datos.

Además, cabe destacar que, a medida que se comienzan a utilizar modelos más complicados en SQL, la metodología de trabajo se vuelve cada vez más tediosa, ya que se deben considerar mayor cantidad de variables y tanto la sintaxis en caso de programar como el uso de la GUI se convierten en tareas no tan triviales como inicialmente podrían parecer.

IV. Conclusión

A lo largo de este trabajo se han analizado distintos aspectos del análisis de datos con Pandas y SQL.

El análisis de datos, como se vio, es un proceso por el cual se busca describir el comportamiento de un grupo de datos, y encontrar sus relaciones, para poder convertir dichos datos en información analizable, reproducible y utilizable.

En materia de análisis de datos, tanto SQL como Python, en particular con su librería Pandas, son dos de las herramientas más utilizadas, ya que ambas proveen algoritmos y funcionalidades que facilitan y optimizan los procesos de análisis de datos previamente mencionados.

Para comenzar el análisis comparativo, se realizó el mismo algoritmo en ambas plataformas. Éste consistió en una serie simple de pasos, pero en la cual se iba variando la cantidad de datos a tomar, desde 10 hasta 10^7 en la mayoría de los casos. Luego se prosiguió a tomar información sobre los tiempos que cada una de las herramientas tardaba en realizar dichos análisis. Como se vio, la clara vencedora fue SQL. Esto se dio porque se encontró que Pandas avanzaba con una complejidad del orden de O(n), es decir, cambiaba linealmente con el tamaño de los datos, mientras que SQL presentó una complejidad del orden de O(log(n)). Este comportamiento puede no parecer tan distinto para valores pequeños de n, lo cual de hecho se vio evidenciado por una clara ventaja de Pandas para valores de n menores a 10^3, pero a medida que n comienza a crecer, la diferencia entre O(n) y O(log(n)) se vuelve más y más evidente y crítica.

Por otro lado, también se realizó un análisis con respecto al uso de la memoria que tanto SQL como Pandas hacían a la hora de efectuar el mismo algoritmo del caso anterior. Nuevamente, se fue variando el tamaño de los datos, y se intentó encontrar relaciones entre dicho tamaño y la cantidad utilizada de memoria. Se pudo observar que, nuevamente, la relación que presentó Pandas fue de carácter lineal, incluso con ciertas tendencias exponenciales. Sin embargo, SQL se mantuvo fiel a su comportamiento logarítmico. Otra vez, para valores pequeños de n Pandas tuvo cierta ventaja, pero no tuvo que crecer demasiado el tamaño del dataset para que el comportamiento logarítmico superara ampliamente en términos de eficiencia al comportamiento lineal. Es por esto que en materia de uso de memoria, SQL es la definitiva ganadora.

Por último, se realizó un análisis comparativo en el proceso de minería de datos. Se explicó que la minería de datos consiste en el primer acercamiento a una serie desconocida de datos, a partir del cual se logran establecer las relaciones más basales y los comportamientos más evidentes que los datos puedan presentar, como así también sus respectivas relaciones. Para esto, se vieron dos enfoques distintos. En el caso de Pandas, se utilizó un enfoque más *hands-on*, es decir, se utilizaron diversos métodos de la clase dataframe para ir viendo de a poco los datos presentados. Yendo desde el caso más general hasta algunos más puntuales, se pueden ir analizando los datos desde un punto de vista más numérico, es decir, cuantitativo. En el caso de SQL, por su parte, se realizó un análisis del tipo cualitativo, con implementaciones de Microsoft de algoritmos extensamente utilizados en minería de datos. Se vio un ejemplo también de regresión lineal, que es uno de estos modelos, y cómo depende del caso qué tan útil sea dicho modelo. Se concluyó finalmente que en términos de minería de datos, Pandas fue la vencedora, ya que presenta las mismas herramientas que SQL y muchas otras más, principalmente provenientes de su utilización integrada en Python, un lenguaje muy versátil y con incontables librerías que quedan, así, a disposición de Pandas.

Finalmente, llega el momento de responder las preguntas planteadas en un principio. ¿Qué herramienta usar? ¿Cuál conviene en cada caso? Si bien pueden parecer dos preguntas similares, las respuestas son muy distintas.

Si se quiere saber qué herramienta utilizar, independientemente de la estructura de los datos, la ganadora definitivamente es SQL. ¿Por qué? Porque es la que mejor cubre los distintos casos que se puedan llegar a presentar. Es cierto que presenta un kit de herramientas mucho más limitado que el de Pandas, pero corre con la invaluable ventaja de la eficiencia. En un contexto donde no se conoce el tamaño de los datos, puede resultar crítica la utilización de la herramienta que mejor haga uso del hardware disponible.

Por otro lado, si la pregunta es sobre cuál herramienta usar en cada caso, la respuesta surge evidentemente a partir de los análisis realizados durante el trabajo. Si los datos que se tienen son de extensión relativamente pequeña, entonces Pandas es el camino ideal. Podrá realizarse un minado de datos mucho más exhaustivo y profundo de lo que podría hacerse con SQL, sin mencionar que, combinado con otras librerías de Python, la visualización y modelización del comportamiento de los datos se convierte en una tarea muy sencilla. Sin embargo, si el dataset que se desea analizar tiene cantidades muy grandes de datos, nuevamente se cae en el caso anterior, en el cual SQL corre con la clara ventaja de la eficiencia, por lo cual cualquier programa hecho con Pandas estará consumiendo recursos que potencialmente podrían ser de vital importancia, o al menos de carácter limitante en caso de no poseerlos.

A modo de conclusión, se considera importante remarcar que, si bien se logró determinar claramente cuál herramienta utilizar en cada caso, la realidad es que ambas se usan en conjunto, porque se complementan muy bien. Como ya se ha estado comentando durante la monografía, la metodología usual consiste en un primer filtrado y reorganización de los datos con SQL, quien cuenta con la velocidad y eficiencia, para luego realizar el análisis detallado de los datos ya filtrados con Pandas. De esta manera, se logran aprovechar las ventajas de ambos lenguajes, sin necesidad de someterse a las correspondientes desventajas.

V. REFERENCIAS

[1] Dinesh Asanka, “Introduction to SQL Server Data Mining”, https://www.sqlshack.com/introduction-to-sql-server-data-mining/, 13/07/2019.

[2] F. Pedregosa, P. Gervais, “Memory Profiler”, PyPI.org, https://pypi.org/project/memory-profiler/.

[3] Facultad de desarrollo y centro instruccional de diseño, *Conductas responsables para la*, https://ori.hhs.gov/education/products/n_illinois_u/datamanagement/datopic.html, Universidad Norteña de Illinois.

[4] M. Steinbach, P. Tan, V. Kumar, “Introduction to Data Mining”, Addison-Wisley, 2005.

[5] Michael Rundell, “Data Mining in Python: A Guide”, https://www.springboard.com/blog/data-mining-python-tutorial/, 13/10/2016.

[6] Microsoft Documentation, SQL Documentation, “Monitor and Troubleshoot Memory Usage”, https://docs.microsoft.com/en-us/sql/relational-databases/in-memory-oltp/monitor-and-troubleshoot-memory-usage?view=sql-server-ver15#bkmk_Monitoring, 17/03/2017.

[7] Python.org, *The Python Standard Library, Generic Operating System Services*, https://docs.python.org/3/library/time.html, sin autor especificado.

[8] Shamoo, Resnik, Conductas Responsables para la Investigación, tercera edición, 2003.

[9] spencer7593, The Moot, StakOverflow, https://stackoverflow.com/questions/11675077/measure-the-time-it-takes-to-execute-a-t-sql-query/11675263, 26/07/2012.

[10] Wes McKinney, “Python for Data Analysis”, O’Reilly, 2nd Edition, 2018.