Reproducción del artículo: Efficient and Distributed Temporal Pattern Mining

1st Esteban Villacorta
Ciencia de la Computación
Universidad de Tecnología e Ingeniería
Lima, Perú
esteban.villacorta@utec.edu.pe

2nd Juan Galvez
Ciencia de la Computación
Universidad de Tecnología e Ingeniería
Lima, Perú
juan.galvez@utec.edu.pe

3rd Jose de Lama
Ciencia de la Computación
Universidad de Tecnología e Ingeniería
Lima, Perú
jose.delama@utec.edu.pe

I. Introducción

Una de las técnicas de minería para extraer patrones de series temporales es *Temporal Pattern Mining (TPM)*. Esta técnica a diferencia del minado secuencial, agrega una dimensión temporal adicional que son intervalos de tiempo para cada patrón haciéndolos más informativos. Sin embargo esta adición hace que la complejidad aumente. Actualmente los enfoques de *TPM* funcionan solo de manera secuencial haciéndolo no escalable para una gran cantidad de datos. Es por eso que Ho, Pedersen y Vu proponen *Distributed Hierarchical Pattern Graph TPM (DHPG-TPM)* [1]. Este modelo se encarga de minar/extraer patrones temporales de manera distribuida utilzando Apache Spark. Con ese modelo la complejidad computacional disminuirá y hará que *TPM* sea escalable para grandes conjunto de datos.

En este informe se replicará dicho trabajo, se discutirá sobre el caso de estudio en base al *dataset* elegido, desarrollo de la replicación y finalmente se expondrán las conclusiones del trabajo realizado.

II. Descripción del caso de estudio

El *dataset* escogido fue extraído de la residencial de energía Net-Zero donde se obtuvieron datos de los sensores por un año [2].

A. Dataset sin procesar

Información del *dataset* antes de procesarlo. En la imagen 1 se aprecia una pequeña parte de la información almacenada. En la tabla I se pueden ver los detalles del *dataset*.

All-Subsystems-minute.csv	
Tamaño	2.07 GB
Temporalidad	Minutos
Número de columnas	358
Número de muestras	518792

TABLE I: Información del *dataset* sin procesar

B. Dataset procesado Dseq

Información del *dataset* después de procesarlo y convertirlo a *D*seq. En la imagen 2 se aprecia una pequeña parte de la información almacenada. En la tabla II se pueden ver los detalles del *dataset*.

Timestamp	TimeStamp_Count	DHW_ClothesWasherColdFlow	DHW_ClothesWasherHotFlow	DHW_DishwasherHotFlow
2013-07-01 04:01:14	1.0	0.0	0.0	0.0
2013-07-01 04:02:14	2.0	0.0	0.0	0.0
2013-07-01 04:03:14	3.0	0.0	0.0	0.0
2013-07-01 04:04:14	4.0	0.0	0.0	0.0
2013-07-01 04:05:14	5.0	0.0	0.0	0.0
2013-07-01 04:06:14	6.0	0.0	0.0	0.0
2013-07-01 04:07:14	7.0	0.0	0.0	0.0
2013-07-01 04:08:14	8.0	0.0	0.0	0.0
2013-07-01 04:09:14	9.0	0.0	0.0	0.0
2013-07-01 04:10:14	10.0	0.0	0.0	0.0
2013-07-01 04:11:14	11.0	0.0	0.0	0.0
2013-07-01 04:12:14	12.0	0.0	0.0	0.0
2013-07-01 04:13:14	13.0	0.0	0.0	0.0
2013-07-01 04:14:14	14.0	0.0	0.0	0.0
2013-07-01 04:15:14	15.0	0.0	0.0	0.0
2013-07-01 04:16:14	16.0	0.0	0.0	0.0
2013-07-01 04:17:14	17.0	0.0	0.0	0.0
2013-07-01 04:18:14	18.0	0.0	0.0	0.0
2013-07-01 04:19:14	19.0	0.0	0.0	0.0
2013-07-01 04:20:14	20.0	0.0	0.0	0.0

Fig. 1: Algunas filas y columnas de *All-Subsystems-minute*

*D*seq	
Número de columnas	3
Número de muestras	16946

TABLE II: Información del *dataset* procesado *Dseq*

C. Dataset procesado Dev

Información del *dataset* después de procesarlo y convertirlo a *D*ev. En la imagen 3 se aprecia una pequeña parte de la información almacenada. En la tabla III se pueden ver los detalles del *dataset*.

*D*ev	
Número de columnas	3
Número de muestras	20

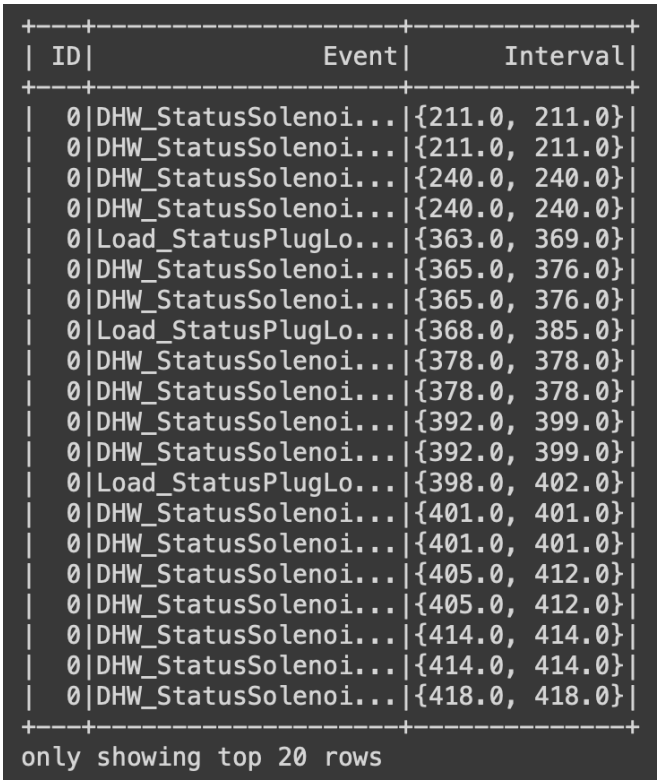
TABLE III: Información del *dataset* procesado *Dev*

III. Arquitectura

Algunas librerías utilizadas en el proyecto fueron las siguientes.

1) pandas
2) pyspark
3) numpy
4) tqdm
5) datetime
6) plotly

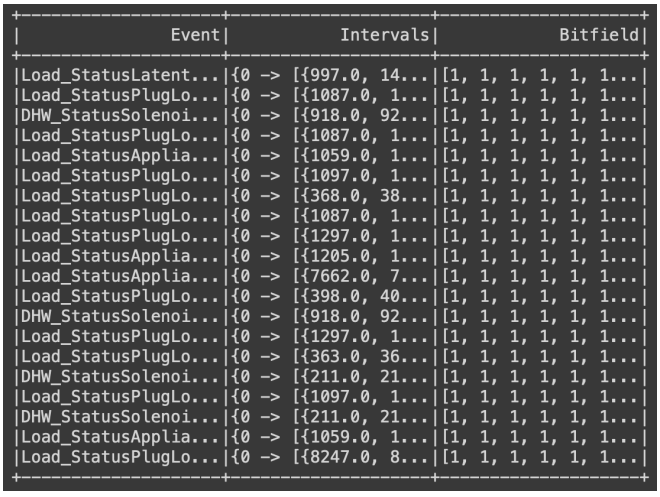
La implementación del algoritmo se realizó puramente en Spark. Se utilizó también NumPy y PlotLy durante una fase previa de exploración.



ID	Event	Interval
0	DHW_StatusSolenoi...	{211.0, 211.0}
0	DHW_StatusSolenoi...	{211.0, 211.0}
0	DHW_StatusSolenoi...	{240.0, 240.0}
0	DHW_StatusSolenoi...	{240.0, 240.0}
0	Load_StatusPlugLo...	{363.0, 369.0}
0	DHW_StatusSolenoi...	{365.0, 376.0}
0	DHW_StatusSolenoi...	{365.0, 376.0}
0	Load_StatusPlugLo...	{368.0, 385.0}
0	DHW_StatusSolenoi...	{378.0, 378.0}
0	DHW_StatusSolenoi...	{378.0, 378.0}
0	DHW_StatusSolenoi...	{392.0, 399.0}
0	DHW_StatusSolenoi...	{392.0, 399.0}
0	Load_StatusPlugLo...	{398.0, 402.0}
0	DHW_StatusSolenoi...	{401.0, 401.0}
0	DHW_StatusSolenoi...	{401.0, 401.0}
0	DHW_StatusSolenoi...	{405.0, 412.0}
0	DHW_StatusSolenoi...	{405.0, 412.0}
0	DHW_StatusSolenoi...	{414.0, 414.0}
0	DHW_StatusSolenoi...	{414.0, 414.0}
0	DHW_StatusSolenoi...	{418.0, 418.0}

only showing top 20 rows

Fig. 2: Algunas filas y columnas de *D*seq



Event	Intervals	Bitfield
Load_StatusLatent...	{0 -> [{997.0, 14...	[1, 1, 1, 1, 1, 1...
Load_StatusPlugLo...	{0 -> [{1087.0, 1...	[1, 1, 1, 1, 1, 1...
DHW_StatusSolenoi...	{0 -> [{918.0, 92...	[1, 1, 1, 1, 1, 1...
Load_StatusPlugLo...	{0 -> [{1087.0, 1...	[1, 1, 1, 1, 1, 1...
Load_StatusApplia...	{0 -> [{1059.0, 1...	[1, 1, 1, 1, 1, 1...
Load_StatusPlugLo...	{0 -> [{1097.0, 1...	[1, 1, 1, 1, 1, 1...
Load_StatusPlugLo...	{0 -> [{368.0, 38...	[1, 1, 1, 1, 1, 1...
Load_StatusPlugLo...	{0 -> [{1087.0, 1...	[1, 1, 1, 1, 1, 1...
Load_StatusPlugLo...	{0 -> [{1297.0, 1...	[1, 1, 1, 1, 1, 1...
Load_StatusApplia...	{0 -> [{1205.0, 1...	[1, 1, 1, 1, 1, 1...
Load_StatusApplia...	{0 -> [{7662.0, 7...	[1, 1, 1, 1, 1, 1...
Load_StatusPlugLo...	{0 -> [{398.0, 40...	[1, 1, 1, 1, 1, 1...
DHW_StatusSolenoi...	{0 -> [{918.0, 92...	[1, 1, 1, 1, 1, 1...
Load_StatusPlugLo...	{0 -> [{1297.0, 1...	[1, 1, 1, 1, 1, 1...
Load_StatusPlugLo...	{0 -> [{363.0, 36...	[1, 1, 1, 1, 1, 1...
DHW_StatusSolenoi...	{0 -> [{211.0, 21...	[1, 1, 1, 1, 1, 1...
Load_StatusPlugLo...	{0 -> [{1097.0, 1...	[1, 1, 1, 1, 1, 1...
DHW_StatusSolenoi...	{0 -> [{211.0, 21...	[1, 1, 1, 1, 1, 1...
Load_StatusApplia...	{0 -> [{1059.0, 1...	[1, 1, 1, 1, 1, 1...
Load_StatusPlugLo...	{0 -> [{8247.0, 8...	[1, 1, 1, 1, 1, 1...

Fig. 3: Algunas filas y columnas de *D*ev

Se realizó también integración del código con Google Drive y Colab. Esto implica que el proyecto fue implementado completamente en un notebook de Colab, que contiene el historial de cambios en el tiempo, junto a los resultados del procesamiento de datos.

IV. Estructura de datos

Debido al formato declarativo de Jupyter, se prefirió mantener la implementación definida por funciones y bloques definidos secuencialmente.

`ProcessEvent` recibe una columna y obtiene, para el evento definido por esta columna, todas las instancias del evento. Una instancia del evento, en este caso, es definido en Spark como:

```
StructType([
    StructField("InstanceId", StringType()),
    StructField("Start", TIME_TYPE),
    StructField("End", TIME_TYPE),
    StructField("EventName", StringType())]
)
```

Esto puede ser posteriormente procesado, resultando en `DSeq`. Nótese que, para la implementación, el ID mappea directamente a la partición. Esto permite una mayor velocidad de procesamiento.

```
StructType([
    StructField("ID", LongType()),
    StructField("Event", StringType()),
    StructField("Interval", IntervalType),
])
```

`DEv` utiliza las particiones definidas en `DSeq`, y define su partición de origen.

```
StructType([
    StructField("Event", StringType()),
    StructField("Intervals", MapType(
        LongType(),
        ArrayType(IntervalType))),
    StructField("Bitfield", ArrayType(
        LongType())),
])
```

A partir de esto, se derivan tanto `\_1Freq` y `\_2Freq`, que se definen por las siguientes estructuras:

```
StructType([
    StructField("Event", StringType()),
    StructField("Intervals", MapType(
        LongType(),
        ArrayType(IntervalType))),
    StructField("Bitfield", ArrayType(
        LongType())),
    StructField("Support", FloatType()),
])

StructType([
    StructField("Event_1", _1Freq),
    StructField("Event_2", _1Freq),
    StructField("Bitfield", ArrayType(
        LongType())),
    StructField("Support", FloatType()),
    StructField("Confidence", FloatType())
])
```

V. Conclusiones

References

[1] N. Ho, V. Ho, T. Pedersen and M. Vu, ”Efficient and Distributed Temporal Pattern Mining”, 2021.

[2] NIST Net-Zero, ”All Subsystems Minute” https://s3.amazonaws.com/nist-netzero/2014-data-files/All-Subsystems-minute.csv

[3] PySpark Documentation https://spark.apache.org/docs/latest/api/python/