Almac. y Proc. Masivo de Datos: Tarea #3

Miércoles, 15 de Noviembre, 2017

Juan Pablo Arévalo

Contents

Introducción

La tarea consistía en trabajar con el conjunto de datos de Yelp, el cual es explicado en profundidad en la Página web de Yelp.

A diferencia de la Tarea #2, el enfoque de esta fue realizar sólo un problema, almacenando el código en un Jupyter Notebook.
Para este problema, se trabajó principalmente con los archivos "business.json" y "review.json".

Para realizar esta tarea, al igual que en la anterior, se trabajó con dos tamaños de dataset, para cada uno de los archivos.
Los archivos originales presentaban un volumen considerablemente grande de datos, por lo que se decidió disminuir el volumen a un numero fijo de elementos. Se optó reducir los conjuntos a un conjunto de 10000 datos (small) y a otro de 30000 datos (medium), para hacer esto, se siguieron los mismos pasos que para la tarea anterior, comenzando por reducir el tamaño del conjunto "business.json" a 10000 y a 30000 utilizando los comandos de consola presentados a continuación:

```
head -n 10000 business.json > business_small.json
head -n 30000 business.json > business_medium.json
```

Luego, se reutilizaron los dos scripts de Python previamente creados para encoger los otros archivos. Ambos scripts funcionan de igual manera, simplemente variando el número de datos que contendrán los nuevos archivos. Los scripts se llaman "shrink_datasets_to_small.py" y "shrink_datasets_to_medium.py" y pueden ser encontrados en la carpeta "Entrega" de la tarea 2.

Los archivos reducidos se almacenaron como listas de objetos JSON, pero los archivos originales eran simplemente objetos JSON no almacenados dentro de una lista. Es por esto que al comienzo del script de respuesta para el problema de esta tarea se implementó y utilizó la función que se presenta a continuación, encargada de identificar el formato del archivo JSON y entregar las lineas en el formato correcto:

```
def get_json_element(line):
    ''' Had to create this function because my shrinked JSON were dumped as a list,
    not as various JSON Objects as in the original files, so this function
    handles both cases (multiple JSON objects and a LIST of JSON Objects) '''
    line = json.loads(line)
    if isinstance(line, list):
        for real_json_object in line:
            yield real_json_object
    else:
        yield line
```

Problema 1

El Problema consistía en encontrar la distribución de estrellas para cada categoría en el set de datos de YELP. El objetivo principal era crear un Jupyter Notebook que señalara los pasos a seguir para lograr esto, el notebook final se puede observar en la siguiente ruta: Notebook Tarea 3.
Cabe destacar que el notebook fue realizado con el set de 30000 datos.
A pesar de que todo se encuentra explicado con detalle en el notebook, esta explicación se encuentra en inglés para que sea entendida universalmente, es por esto que, en este informe, se explicará nuevamente el proceso en español.
Por motivos de simplicidad, no se presentarán ni explicarán todos los códigos en este informe, ya que se encuentran explicados en el Notebook. A pesar de esto, se utilizarán fragmentos de los códigos para clarificar el proceso y solución del problema.

Se comenzó por dividir el problema en 3 subtareas:

1. Crear un archivo CSV que contenga todos los reviews y las categorías correspondientes a sus respectivos businessess.
2. Utilizando Map-Reduce, obtener la distribución de estrellas para cada categoría, almacenándola en un CSV.
3. Utilizando la librería MatPlotLib, graficar histogramas mostrando la distribución de estrellas para cada categoría.

Para el primer punto, se comenzó por leer el archivo business.json, almacenando en un diccionario las categorías correspondientes, para luego leer el archivo reviews.json y guardar en un CSV cada review junto con sus categorías correspondientes y las estrellas asignadas, siguiendo el siguiente formato:

```
review_id_1;category_1;stars
review_id_1;category_2;stars
review_id_2;category_1;stars
...
```

El código encargado de realizar este procedimiento se puede ver a continuación:

```
BUSINESS_CATEGORIES = {}
business = sys.argv[1]
reviews = sys.argv[2]
out_name = sys.argv[3]
files = [business, reviews]
out_file = open(out_name, "w")
for f in files:
    opened_file = open(f,"r")
    for line in opened_file:
        for l in get_json_element(line):
            if 'categories' not in l:
                if l['business_id'] in BUSINESS_CATEGORIES:
                    categories = BUSINESS_CATEGORIES[l['business_id']]
                    for category in categories:
                        review_id = l['review_id']
                        stars = str(l['stars'])
                        out_file.write(review_id+';'+category+';'+stars+ '\n')
            else:
                BUSINESS_CATEGORIES[l['business_id']] = l['categories']
out_file.close()
```

Para llamar a este script, se debe ejecutar el siguiente comando, entregando la ruta de los archivos JSON de business y reviews, así como también la ruta del archivo CSV que se generará.

```
python generate_csv.py business_medium.json review_medium.json
    reviews_with_stars_by_category.csv
```

Una vez generado el CSV conteniendo los reviews con sus respectivas categorías procedemos al paso 2, donde con la ayuda de la librería de Map-Reduce MRJob obtenemos la distribución de estrellas para cada categoría. Para comenzar se ejecuta un mapper encargado de convertir el número de estrellas en cada review a un formato estándar, representado por una lista, donde cada índice representa la cantidad i de estrellas que se asignaron al review. Es decir, si el review tiene 4 estrellas asignadas, se mappeará esto a algo de la forma : [0, 0, 0, 1, 0].
Esto luego se entrega al siguiente paso junto con la categoría correspondiente al review.
El código de este mapper se puede ver a continuación:

```
def mapper_user_reviews_by_cat(self, _, line):
    line = line.split(';')
    stars = int(line[2])
    category = line[1]
    star_array = [0,0,0,0,0]
    star_array[stars-1] = 1
    yield [category, star_array]
```

El reducer siguiente se encarga de juntar todas las estrellas entregadas a cada categoría en forma de lista, entregando algo de la siguiente forma: Categoría, [[0,0,0,1,0], [0,1,0,0,0], [0,0,0,0,1], ...]
El código del reducer se puede ver a continuación:

```
def ratings_per_category_reducer(self, key, values):
        yield key, list(values)
```

Por ultimo, el último reducer se encarga de sumar el número de estrellas obtenido en cada índice para cada categoría. Esto se retorna en formato CSV utilizando el protocolo CSV de la librería mr3px, entregando algo de la siguiente forma: "Categoría;0;1;0;1;1"
El código se puede ver a continuación:

```
def total_ratings_per_category_reducer(self, key, values):
        for value in values:
            category = key
            stars = [str(sum(x)) for x in zip(*value)]
            yield "STARS", [category] + stars
```

Estas líneas retornadas deberán guardarse en un archivo CSV, esto se hace al momento de ejecutar el script en consola, como se puede ver a continuación:

```
python mr_get_star_distribution.py reviews_with_stars_by_category.csv >>
    categories_with_stars.csv
```

Una vez obtenido este CSV, sólo queda graficar los datos en forma de histograma para cada categoría.
A modo de realizar esto de manera un póco más amigable para el usuario, se agregó la opción de mostrar todas las categorías, mostrarlas una por una, o mostrar categorías que satisfagan un criterio de búsqueda ingresado por el usuario.
Esta funcionalidad se puede ver más en detalle al leer el Notebook.
El código correspondiente a la generación del histograma se puede ver en el Notebook, se utilizó principalmente Matplotlib para realizar los gráficos. Un ejemplo de los resultados obtenidos se puede ver en la Figura 1.

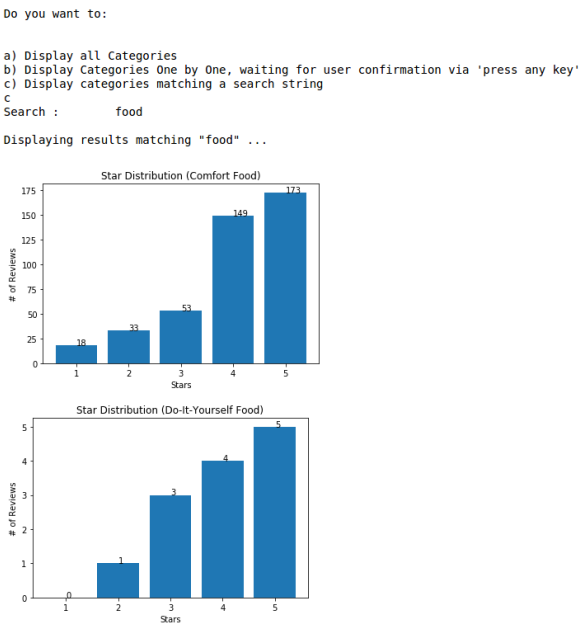

Figure 1: Histogramas por categoría (utilizando el buscador).

En el notebook se puede ejecutar el proceso para los archivos de 10000 y 30000 datos.
Los resultados no varían mucho, principalmente varía el número de categorías totales, pero la distribución de estrellas es similar para las categorías, esto se puede revisar comparando los gráficos obtenidos con el Notebook.

Bonus

Se solicitó utilizar la librería Lightning para Jupyter Notebook, la cual genera histográmas y gráficos más ordenados y presentables. Esta librería sin embargo, presenta varios errores a la hora de intentar graficar varias categorías a la vez, ya que al ser un servicio externo permite mostrar los gráficos sólamente de a uno. Es por esto que se implementó la funcionalidad de graficar con lightning por separado al programa principal, entregándole al usuario una lista con todas las categorías encontradas, para que este ingrese el número correspondiente y luego presentar el histograma de dicha categoría, por medio de la librería Lightning.

El resultado se puede ver en la Figura 2.

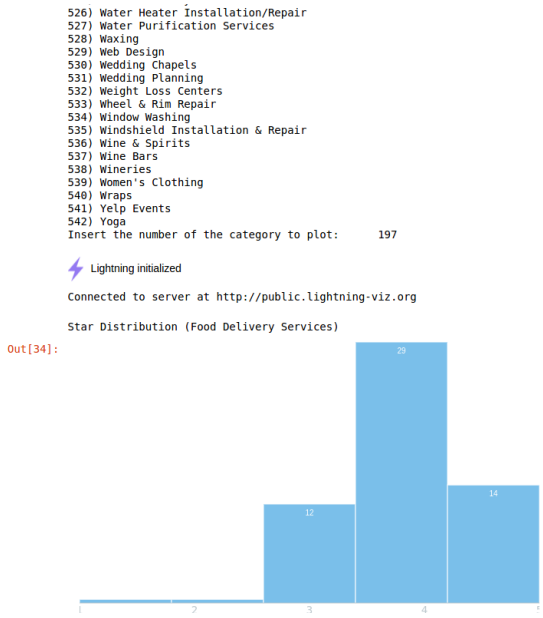

Figure 2: Histogramas por categoría (utilizando el buscador).

A modo de concluir este ítem de Bonus, no fue muy agradable trabajar con Lightning, la documentación es muy reducida, presenta comportamiento errático y tiene pocas opciones de customización.
No se logró agregar títulos, leyendas o ejes a los gráficos, resultando en un histograma que, si bien se ve más sofisticado, presenta falta de información y tomó mucho tiempo en desarrollar intentando customizarlo con mayor profundidad.

Conclusiones

A modo de concluir, la tarea fue muy útil para comprender el funcionamiento de la Jupyter Notebooks, una herramienta muy útil para escribir código y, lo más importante, compartirlo y explicarlo.
Jupyter presenta un gran potencial a la hora de enseñar, explicar y compartir código, ya que tiene un grán numero de opciones, tanto de código como de escritura, además de permitir que los notebooks sean descargados y compartidos de manera fácil.

Fue muy útil aprender de Jupyter mientras se ejercitaba el uso de la librería MRJob, la cual comenzamos a utilizar en la primera tarea, se pudo practicar bastante y se mejoró mucho la calidad y el empleo de dicha librería.

Cabe destacar que los tiempos de ejecución fueron bastante decentes (tan sólo unos segundos), tanto con el set de 30000 como con el set de 10000, lo que nos comprueba la eficacia de esta herramienta a la hora de realizar procesos de agrupación y segmentación de datos en un set de datos muy poblado.

La tarea fue una gran fuente de aprendizaje y práctica, se logró analizar conjuntos de volumenes considerables de datos, encontrar estadísticas de interés y practicar con la librería de MRJob y la herramienta Jupyter Notebooks.

A lo largo del desarrollo de la tarea, principalmente se investigó en el foro de StackOverflow y en la Documentación de MRJob, así como también en la Documentación de Jupyter. Además, también se utilizó las siguientes fuentes para comprender mejor el funcionamiento de MRJob, Lightning, Matplotlib y Jupyter:

https://blog.matthewrathbone.com/2016/02/09/python-tutorial.html
http://aimotion.blogspot.cl/2012/08/introduction-to-recommendations-with.html
http://mrjob.readthedocs.io/en/latest/guides/writing-mrjobs.html
https://www.datacamp.com/community/tutorials/tutorial-jupyter-notebook
https://github.com/lightning-viz/lightning-example-notebooks/blob/master/plots/histogram.ipynb
https://matplotlib.org/contents.html