Almac. y Proc. Masivo de Datos: Tarea #2

Martes, 10 de Octubre, 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.

Se trabajó principalmente con los archivos "business.json", "review.json" y "users.json", pero aún así se mantuvieron todos los archivos originales del dataset.

Para realizar esta tarea, 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 comenzó reduciendo 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 crearon dos scripts de Python 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" del proyecto.

En primer lugar, se lee el archivo "business_small.json"/"business_medium.json" y se guardan los ID de los negocios que contiene en una lista. Luego, se encoge el dataset "review.json", preocupándose de utilizar sólo reviews a negocios cuyo ID esté presente en la lista obtenida del archivo de businesses.

Luego, se almacenan también los ID de los reviews del nuevo dataset reducido, para pasar a encoger el dataset "user.json", preocupándose de utilizar sólo usuarios cuyo ID haya realizado alguno de los reviews del set de reviews encogido.

Finalmente, se procede a encoger los sets restantes ("photos.json", "tip.json", "checkin.json") preocupándose de que estén asociados a un negocio en la lista reducida de businesses.

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 de cada script 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 el review más único del conjunto de datos. Para que un review sea considerado como el más único, este debe tener la mayor cantidad de palabras usadas una sola vez en todo el set de reviews. Estas palabras no deben considerar signos de puntuación.

El script debe ser llamado de la siguiente manera:

```
python 01_unique_review.py review_small.json
```

Para resolver este problema, se comenzó diseñando un mapper que limpiara los reviews de todo signo de puntuación, para luego entregar cada palabra por separado con un valor de uno. Al mismo tiempo, se registró en un diccionario el review correspondiente a cada palabra.

```
WORDS = {}
def mapper_words(self, _, line):
        for l in get_json_element(line):
            review = re.sub(r'[^\w]', ' ', l['text'])
            for word in review.strip().split():
                WORDS[word.lower()] = [l['review_id'],review]
                yield [word.lower(), 1]
```

Luego, El primer reducer sumaba todas las palabras iguales, obteniendo una lista con cada palabra y un contador que indica el número de veces que se repite en el dataset.

```
def reducer(self, key, values):
        yield ['MAX',[sum(values),key]]
```

Este reducer entregaba los datos a un nuevo mapper que se encargaba de dejar pasar sólamente las palabras cuyo conteo final era de 1 (es decir, que aparecían una sola vez en todo el dataset). Pero en lugar de pasar las palabras, pasaba el review correspondiente a esta (con ayuda del diccionario creado anteriormente), entregando finalmente los reviews, acompañados de un valor de uno.

```
def mapper_unique_word_in_review(self, stat, values):
        if values[0] == 1:
            yield [WORDS[values[1]], 1]
```

El reducer siguiente, al igual que el anterior suma todos los reviews que se repiten, manteniendo un contador que representa el número de palabras únicas en el review.

```
def reducer2(self, key, values):
        yield ['MOST UNIQUE REVIEW',[sum(values),key]]
```

Problema 1 continúa en la siguiente página...

Por ultimo, el último reducer obtiene el máximo de los resultados entregados anteriormente, indicando el ID del review con palabras más únicas, su texto y por último el número de palabras únicas que contiene.

```
def reduce_uniquest_review(self, key, values):
        yield [key, max(values)]
```

El resultado se presenta de la siguiente manera:

```
Running step 1 of 3...
Running step 2 of 3...
Running step 3 of 3...
Streaming final output from /tmp/01_unique_review.huarn.20171010.151551.839154/output...
"MOST UNIQUE REVIEW"    [85, ["n1-9tumXHN6w3HJTUti4Fg", "Angeregt durch die Cover Story im
    aktuellen Hannover geht aus und abgeschreckt durch die miserable Performance des
    Castello  http  www restaurant kritik de bewertungen 230653 comments  in Hannover  dass
     denselben Betreibern geh rt  waren meine Frau ..."]]
```

Un problema recurrente a lo largo de la realización de la tarea fue intentar pasar más de un parámetro por los pasos de MRJob, es por esto que se creó el diccionario de palabras (que indica el review al que pertenecen), ya que al pasar como parámetro el review junto con la palabra, la suma del generador no funcionaba.

Dejando esos problemas de lado, el problema logró responderse y los reviews encontrados son efectivamente los más únicos del set.

El programa se ejecutó con los siguientes resultados (acortados) y tiempos de ejecución:

10000 datos:

```
"MOST UNIQUE REVIEW"    [85, ["n1-9tumXHN6w3HJTUti4Fg", "Angeregt durch die Cover Story im
    aktuellen Hannover geht aus und abgeschreckt durch die miserable Performance des
    Castello  http  www restaurant kritik de bewertungen 230653 comments  in Hannover  dass
     denselben Betreibern geh rt  waren meine Frau ..."]]

real 0m16.193s
user 0m16.312s
sys  0m0.384s
```

30000 datos:

```
"MOST UNIQUE REVIEW"    [111, ["tP_Az0FQl12y2x_K0ca9JQ", "Wenn ich innerhalb von gut 1
    Jahren vier Mal in einem Museum war  f r das ich eine Zugfahrt von rund 1  Stunden
    hinter mich bringen muss  was f r mich nicht wirklich eine weite Strecke ist  dann muss
     das Museum einfach gut sein  Und so ist es auch mit dem Landesmuseum ..."]]

real 0m43.878s
user 0m45.144s
sys  0m0.796s
```

Problema 2

El Problema consistía en encontrar los usuarios más similares en base a los negocios visitados, esto comparando los usuarios que realizaron reviews a los mismos negocios.
El índice utilizado para evaluar la similaridad entre los usuarios fue el Índice de Jaccard.

El script debe ser llamado de la siguiente manera:

```
    python 02_similar_users.py review_small.json
```

Para resolver este problema, se comenzó diseñando un mapper que entregara el ID de cada usuario y el ID del negocio evaluado, para cada review en el set de datos entregado.

```
def mapper_user_business(self, _, line):
        for l in get_json_element(line):
            yield l['user_id'], l['business_id']
```

Luego, El primer reducer juntaba todos los negocios evaluados por cada usuario y los entregaba como una lista asociada al ID del usuario.

```
def reducer_businessess_by_user(self, user_id, business_id):
        yield user_id, list(business_id)
```

El mapper siguiente se encargaba de entregar como valor una lista que en el índice 0 entregará el ID del usuario y en el índice 1 los negocios evaluados por este.

```
def mapper_business_reviewed_by_user(self, user_id, business_id_list):
        yield 'businessess_reviewed_by_user', [user_id, business_id_list]
```

El reducer que sigue se encarga de realizar la combinatoria para todo par de usuarios posible en el set de datos, para proceder a calcular el coeficiente de jaccard corresondiente en base al total de negocios que ambos usuarios evaluaron (intersección) y al total de negocios que cada uno evaluó por su parte (unión). Finalmente se entregan los resultados para todos los pares de usuarios cuyo coeficiente Jaccard es mayor a 0.5

```
def reducer_jaccard_similarity(self, _, users_with_businessess):
        for combination in itertools.combinations(users_with_businessess, 2):
            first_user, second_user = combination
            first_user_id, first_users_businessess = first_user
            second_user_id, second_user_businessess = second_user
            same_businessess = (set(first_users_businessess)
                                & set(second_user_businessess))
                                # Set removes duplicates
            intersection = len(same_businessess)
            all_businessess = (set(first_users_businessess
                                + second_user_businessess))
                                # Set removes duplicates
            union = len(all_businessess)
            jaccard_coef = float(intersection)/float(union)
            if jaccard_coef > 0.5:
                yield [first_user_id, second_user_id], jaccard_coef
```

El resultado final presenta los pares de usuarios (sus IDs) junto con su índice de Jaccard correspondiente, como se muestra a continuación:

```
Running step 1 of 3...
Running step 2 of 3...
Running step 3 of 3...
Streaming final output from /tmp/02_similar_users.huarn.20171010.162959.109357/output...
["tG24pgYOAyEONNhgmhVO-Q", "z42tHgfcWXNA60x0Japutg"]    1.0
["tGx9-4LpdahotPuNuVfNVQ", "xHvXI-rNuk52jehqnQS0eg"]    1.0
...
```

Un problema en este conjunto de datos es que muchos usuarios realizaron un review a sólamente un negocio, generándose muchos índices de Jaccard iguales a 1.0, y pocos datos donde el índice probara significancia.

El programa se ejecutó con los siguientes resultados (acortados) y tiempos de ejecución:

10000 datos:

```
["tG24pgYOAyEONNhgmhVO-Q", "z42tHgfcWXNA60x0Japutg"]    1.0
["tGx9-4LpdahotPuNuVfNVQ", "xHvXI-rNuk52jehqnQS0eg"]    1.0
...

real 1m8.027s
user 1m4.932s
sys  0m1.040s
```

30000 datos:

```
["zs7P3o9stNXPc8723alvHg", "zv9BgxvwG_h_M-WdxyGtoQ"]    1.0
["zs7P3o9stNXPc8723alvHg", "zxEnZtNbS8RWSSZwcXPPbw"]    0.6666666666666666
...

real 8m22.575s
user 7m52.576s
sys  0m9.028s
```

Problema 3

El Problema consistía en encontrar los usuarios con más reviews para cada categoría de negocios.

Este problema implicaba trabajar con dos sets de datos simultáneamente, por una parte se necesitaba el set de reviews para ver las evaluaciones y por otra parte era necesario el set de business para ver las categorías de los negocios involucrados en cada review. Para hacer esto, el script se debe llamar de la siguiente manera (es importante el orden de los archivos de entrada):

```
python 03_top_reviewers_by_category.py business_small.json review_small.json
```

El primer mapper se encargaba de almacenar las categorías de cada business en un diccionario, para que luego, al leer el archivo de reviews, se les asigne a estos las categorías asociadas al negocio en cuestión, para luego entregar al reducer un par usuario-categoría acompañado de un valor de uno, para luego juntar el número de pares usuario-categoría que se repiten (es decir, el número de reviews por usuario a una determinada categoría).
También se crea un diccionario para almacenar qué usuario lleva actualmente la mayor cantidad de reviews en cada categoría, el cual por el momento se rellena con vacíos.
Se entrega finalmente un valor de ['end', -1] para indicar el final de la lista de usuarios con sus respectivos números de reviews por categoría.

```
BUSINESS_CATEGORIES = {}
MAX_REVIEWS_BY_CAT = {}
def mapper_user_reviews_by_cat(self, _, line):
        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:
                        if category not in MAX_REVIEWS_BY_CAT:
                            MAX_REVIEWS_BY_CAT[category] = [0,'none']
                        yield [l['user_id']+';'+category, 1]
            else:
                BUSINESS_CATEGORIES[l['business_id']] = l['categories']
        yield ['end',-1]
```

Luego, El primer reducer juntaba todos los pares usuario-categoría, indicando la cantidad de reviews de cada usuario para una determinada categoría

```
def reducer(self, key, values):
        yield ['MAX',tuple([sum(values),key])]
```

El último reducer, lee línea por linea los usuarios y sus reviews por categoría y va actualizando el diccionario de MAX_REVIEWS_BY_CAT con el usuario que actualmente presenta la amyor cantidad de reviews en la categoría en cuestión. Una vez que se lee el último elemento (['end', .1]) se procede a retornar los elementos del diccionario MAX_REVIEWS_BY_CAT, es decir, los usuarios que presentan mayor número de reviews para todas las categorías.

```
def reduce_max_reviews_by_category(self, stat, values):
        if values[1] != 'end':
            total_reviews = values[0]
            user_id, category = values[1].split(';')
            if total_reviews > MAX_REVIEWS_BY_CAT[category][0]:
                MAX_REVIEWS_BY_CAT[category] = [values[0], user_id]
        else:
            for k, v in MAX_REVIEWS_BY_CAT.items():
                yield ['MORE REVIEWS BY CATEGORY', [k, v[0], v[1]]]
```

El resultado final presenta todas las categorías involucradas en los reviews analizados, acompañadas del número de reviews realizados por el usuario que presentó más reviews, y seguido por el ID de dicho usuario.

```
Running step 1 of 2...
Running step 2 of 2...
Streaming final output from /tmp/03_top_reviewers_by_category.huarn.20171010.165844.094838/
    output...
"MORE REVIEWS BY CATEGORY"      ["Truck Rental", 1, "1C-gULkagrtWdsXMawzCfw"]
"MORE REVIEWS BY CATEGORY"      ["Drugstores", 3, "LfKyD_XaTnfKoKujRZ1Wug"]
"MORE REVIEWS BY CATEGORY"      ["Pool & Hot Tub Service", 1, "0w05__BsyugKvRhIvUEnww"]
"MORE REVIEWS BY CATEGORY"      ["Buffets", 1, "--PJ1FuEoTEo-3Cxf_izRg"]
"MORE REVIEWS BY CATEGORY"      ["Auto Insurance", 1, "-Z3q48YMSfMWAP6U2RqnLg"]
"MORE REVIEWS BY CATEGORY"      ["Child Care & Day Care", 1, "1717bIJc6zcuIY_QwpMrSw"]
"MORE REVIEWS BY CATEGORY"      ["Creperies", 1, "-9b4s874f_CnznTu4JorRg"]
"MORE REVIEWS BY CATEGORY"      ["Canadian (New)", 2, "JnPIjvC0cmooNDfsa9BmXg"]
...
```

Al igual que en el primer problema, a pesar de numerosos intentos para pasar las categorías como otro parámetro o intentar hacer una especie de "join" entre ambos sets de datos, se finalizó por utilizar un diccionario global para almacenar las categorías y los máximos reviewers.

El programa se ejecutó con los siguientes resultados (acortados) y tiempos de ejecución:

10000 datos:

```
"MORE REVIEWS BY CATEGORY"      ["Juice Bars & Smoothies", 1, "-2gOxVWcnBr5DclrrsWXCA"]
"MORE REVIEWS BY CATEGORY"      ["Pet Boarding/Pet Sitting", 1, "0B1aRg1_k_5WySlWkhCzCw"]
"MORE REVIEWS BY CATEGORY"      ["Grocery", 1, "0uALWZDD4Hu3_8fykVbL7w"]
"MORE REVIEWS BY CATEGORY"      ["Interval Training Gyms", 1, "2Vf-aWYIyDkNOsHPm-AHNg"]
"MORE REVIEWS BY CATEGORY"      ["Hair Removal", 1, "0KueQxXgax9GlbZ0UNaGlg"]
"MORE REVIEWS BY CATEGORY"      ["Buffets", 1, "FPLGHrDuZnl3rJnnM-NMcQ"]
"MORE REVIEWS BY CATEGORY"      ["Motorsport Vehicle Repairs", 1, "CfNZau7w35MOlPK3Ch1GGA"]
"MORE REVIEWS BY CATEGORY"      ["Community Service/Non-Profit", 1, "0-wo7wKN_Rp0Xs6cBMEDJQ"]
"MORE REVIEWS BY CATEGORY"      ["Bowling", 1, "-Vu7L3U7-kxDyY1VHxw3zw"]
"MORE REVIEWS BY CATEGORY"      ["Restaurants", 4, "VOUnXOlll52vwQcJK1tm8A"]
...

real 0m1.964s
user 0m1.864s
sys  0m0.084s
```

30000 datos:

```
"MORE REVIEWS BY CATEGORY"      ["Juice Bars & Smoothies", 1, "-2gOxVWcnBr5DclrrsWXCA"]
"MORE REVIEWS BY CATEGORY"      ["Pet Boarding/Pet Sitting", 2, "cM5nBvgS1Lmmto_dCfwY3Q"]
"MORE REVIEWS BY CATEGORY"      ["Grocery", 2, "L8cvMZLWmM4qWTl9SkxJ8g"]
"MORE REVIEWS BY CATEGORY"      ["Interval Training Gyms", 1, "-XoCb6sUMa7NoFayUW0FlA"]
"MORE REVIEWS BY CATEGORY"      ["Hair Removal", 1, "--ERQVpqAAoi262TTbLVzQ"]
"MORE REVIEWS BY CATEGORY"      ["Dog Parks", 1, "1fNQRju9gmoCEvbPQBSo7w"]
"MORE REVIEWS BY CATEGORY"      ["Adult Education", 1, "0_C6EjUP5_Jf-4Jckyz4ZA"]
"MORE REVIEWS BY CATEGORY"      ["Motorsport Vehicle Repairs", 1, "CfNZau7w35MOlPK3Ch1GGA"]
"MORE REVIEWS BY CATEGORY"      ["Community Service/Non-Profit", 1, "-BJTaybCScNgY73ph1TE8Q"]
"MORE REVIEWS BY CATEGORY"      ["Bowling", 1, "1De45K4AHpVTovLFlLwVlQ"]
"MORE REVIEWS BY CATEGORY"      ["Restaurants", 12, "CxDOIDnH8gp9KXzpBHJYXw"]
...

real 0m4.874s
user 0m4.720s
sys  0m0.156s
```

El problema también especificaba filtrar a los usuarios por la ponderación de sus votos, esto al no estar explicado de manera muy clara, se interpretó como que se buscaba elegir a los usuarios con mayor número de votos en cada categoría.
Para esto, se creó otro script que funciona de manera análoga al anterior, pero en lugar de utilizar un 1 en el primer mapper, ingresa el número de votos totales para cada review ('useful', 'cool', 'funny'). para luego sumar en base a esos valores y encontrar el usuario con más votos en cada categoría.
El nuevo script se ejecuta de la siguiente forma:

```
python 03_top_reviewers_by_category_and_votes.py business_small.json review_small.json
```

El nuevo programa se ejecutó con los siguientes resultados (acortados) y tiempos de ejecución:

10000 datos:

```
"MORE REVIEWS BY CATEGORY"      ["Juice Bars & Smoothies", 65, "Hi10sGSZNxQH3NLyWSZ1oA"]
"MORE REVIEWS BY CATEGORY"      ["Pet Boarding/Pet Sitting", 3, "62GNFh5FySkA3MbrQmnqvg"]
"MORE REVIEWS BY CATEGORY"      ["Grocery", 13, "HXOfPXPMvdXNpQ_l0RRjKw"]
"MORE REVIEWS BY CATEGORY"      ["Interval Training Gyms", 2, "FSr1CdDnHilQ2forw4buLw"]
"MORE REVIEWS BY CATEGORY"      ["Hair Removal", 27, "bLbSNkLggFnqwNNzzq-Ijw"]
"MORE REVIEWS BY CATEGORY"      ["Buffets", 3, "RSQgTAa-6Pne8nIlSJoKuQ"]
"MORE REVIEWS BY CATEGORY"      ["Motorsport Vehicle Repairs", 2, "U2DXAEeMAHQbZNDznULIUQ"]
"MORE REVIEWS BY CATEGORY"      ["Community Service/Non-Profit", 2, "OEXrsx1-tCXu6vr-WhlL_w"]
"MORE REVIEWS BY CATEGORY"      ["Bowling", 19, "Ic6-gs1_FjrWGx6JIr95Mw"]
"MORE REVIEWS BY CATEGORY"      ["Restaurants", 167, "Fv0e9RIV9jw5TX3ctA1WbA"]
...

real 0m2.108s
user 0m1.992s
sys  0m0.100s
```

30000 datos:

```
"MORE REVIEWS BY CATEGORY"      ["Juice Bars & Smoothies", 65, "Hi10sGSZNxQH3NLyWSZ1oA"]
"MORE REVIEWS BY CATEGORY"      ["Pet Boarding/Pet Sitting", 8, "NkAS3aHj1wbYuQqdruqYMA"]
"MORE REVIEWS BY CATEGORY"      ["Grocery", 50, "Q6iiwjxWCfesMNz2sJYHZw"]
"MORE REVIEWS BY CATEGORY"      ["Interval Training Gyms", 22, "Oz0qgUdME7LqXxJEIZq0Pw"]
"MORE REVIEWS BY CATEGORY"      ["Hair Removal", 27, "bLbSNkLggFnqwNNzzq-Ijw"]
"MORE REVIEWS BY CATEGORY"      ["Dog Parks", 14, "EXKDLuYoZUKUNNNYA7TlGQ"]
"MORE REVIEWS BY CATEGORY"      ["Adult Education", 12, "Isf8G6HPbNqEisKDjmUlbw"]
"MORE REVIEWS BY CATEGORY"      ["Motorsport Vehicle Repairs", 2, "U2DXAEeMAHQbZNDznULIUQ"]
"MORE REVIEWS BY CATEGORY"      ["Community Service/Non-Profit", 5, "13sD6TO71SdkB_Xcn275Ug"]
"MORE REVIEWS BY CATEGORY"      ["Bowling", 26, "NfU0zDaTMEQ4-X9dbQWd9A"]
"MORE REVIEWS BY CATEGORY"      ["Restaurants", 257, "8DEyKVyplnOcSKx39vatbg"]
...

real 0m5.218s
user 0m4.988s
sys  0m0.232s
```

Problema 4

El Problema consistía en encontrar los usuarios más similares en base a los negocios visitados y la evaluación (estrellas) que se asignó a estos, esto comparando los usuarios que realizaron reviews a los mismos negocios. El diseño de la solución es bastante similar a la solución del Problema 2, pero con la diferencia de que en lugar de buscar el índice de Jaccard, se normaliza las estrellas entregadas por cada usuario al negocio.
El script debe ser llamado de la siguiente manera:

```
python 04_similar_users_by_stars.py review_small.json
```

Para resolver este problema, se comenzó diseñando un mapper que entregara el ID de cada usuario y el ID del negocio evaluado, para cada review en el set de datos entregado. Además, se almacena en un diccionario el número de estrellas normalizadas que entregó cada usuario a los respectivos negocios.

```
REVIEW_STARS = {}
def mapper_user_business(self, _, line):
    for l in get_json_element(line):
        normalized_stars = float(l['stars'])/5
        REVIEW_STARS[l['user_id']+';'+l['business_id']] = normalized_stars
        yield l['user_id'], l['business_id']
```

Luego, El primer reducer juntaba todos los negocios evaluados por cada usuario y los entregaba como una lista asociada al ID del usuario.

```
def reducer_businessess_by_user(self, user_id, business_id):
    yield user_id, list(business_id)
```

El mapper siguiente se encargaba de entregar como valor una lista que en el índice 0 entregara el ID del usuario y en el índice 1 los negocios evaluados por este.

```
def mapper_business_reviewed_by_user(self, user_id, business_id_list):
    yield 'businessess_reviewed_by_user', [user_id, business_id_list]
```

El reducer que sigue se encarga de realizar la combinatoria para todo par de usuarios posible en el set de datos, para proceder a calcular el coeficiente de "estrella" corresondiente en base al total de negocios que ambos usuarios evaluaron (intersección) y al número estrellas que cada uno asignó a cada negocio del conjunto. Finalmente se entregan los resultados para todos los pares de usuarios cuyo coeficiente estrella es mayor a 0.5

Problema 4 continúa en la siguiente página...

```
def reducer_star_similarity(self, _, users_with_businessess):
        for combination in itertools.combinations(users_with_businessess, 2):
            first_user, second_user = combination
            first_user_id, first_users_businessess = first_user
            second_user_id, second_user_businessess = second_user
            same_businessess = (set(first_users_businessess)
                                & set(second_user_businessess))
                                # Set removes duplicates
            numerator = 0
            denominator_a = 0
            denominator_b = 0
            for business in same_businessess:
                numerator += REVIEW_STARS[first_user_id + ';' + business] *
                             REVIEW_STARS[second_user_id + ';' + business]
                denominator_a += REVIEW_STARS[first_user_id + ';' + business]**2
                denominator_b += REVIEW_STARS[second_user_id + ';' + business]**2
            star_coef = float(numerator) / (denominator_a**(1/2) * denominator_b**(1/2))
            if star_coef > 0.5:
                yield [first_user_id, second_user_id], star_coef
```

El resultado final presenta los pares de usuarios (sus IDs) junto con su coeficiente estrella correspondiente, como se muestra a continuación:

```
Running step 1 of 3...
Running step 2 of 3...
Running step 3 of 3...
Streaming final output from /tmp/04_similar_users_by_stars.huarn.20171010.182434.320224/
    output...
["zg6lm7DLKJArzFxLDILhwQ", "zjok-SXkvRK7ECK7fZ6mJA"]    0.6
["zhezzCPq7EvC2JvIkAxFxg", "zjok-SXkvRK7ECK7fZ6mJA"]    1.0
["ziiMYejVJAjbkRjSeAzrbQ", "zlIqCecpj5s3EVUNNFdt-A"]    0.8
["zj6WkBWlqpkuFYqcYWmE_w", "zoIMcPKKhm8T5EU0y0RxFg"]    1.0
["zl1FHkee4MInRVUhQKus_A", "zoY8tlqhMj3K_LwfH8D-_g"]    0.6400000000000001

...
```

El programa se ejecutó con los siguientes resultados (acortados) y tiempos de ejecución:

10000 datos:

```
["zg6lm7DLKJArzFxLDILhwQ", "zjok-SXkvRK7ECK7fZ6mJA"]    0.6
["zhezzCPq7EvC2JvIkAxFxg", "zjok-SXkvRK7ECK7fZ6mJA"]    1.0
["ziiMYejVJAjbkRjSeAzrbQ", "zlIqCecpj5s3EVUNNFdt-A"]    0.8
["zj6WkBWlqpkuFYqcYWmE_w", "zoIMcPKKhm8T5EU0y0RxFg"]    1.0
["zl1FHkee4MInRVUhQKus_A", "zoY8tlqhMj3K_LwfH8D-_g"]    0.6400000000000001
...

real 0m59.094s
user 0m56.548s
sys  0m0.832s
```

30000 datos:

```
["zj6WkBWlqpkuFYqcYWmE_w", "zoIMcPKKhm8T5EU0y0RxFg"]    1.0
["zkkYfWKwvp1wFHdwOMSsmw", "zwIsY_-GVXpffLTveyux3g"]    0.8
["zl1FHkee4MInRVUhQKus_A", "zoY8tlqhMj3K_LwfH8D-_g"]    0.6400000000000001
["zl9m7BzuAZgWtiuPpvNuzw", "zpqLuqKk-mZZN3fm6K8nvw"]    0.6400000000000001
["zl9m7BzuAZgWtiuPpvNuzw", "zs9gMwoKwHtp0PalclvV8A"]    0.8
["zlXh3Ij29Xm6TuREOY3RQw", "zuRNDHB0TPEVDMec-pdk0g"]    0.6400000000000001
...

real 8m31.608s
user 7m59.604s
sys  0m9.488s
```

En comparación al Problema 2, el índice estrella presenta resultados mucho más específicos para la similitud entre usuarios. El problema 2 se concentraba en encontrar similitud en los lugares visitados, lo cual, como pudimos ver, presentaba problemas a la hora de encontrar pares de usuarios que visitaban ún solo negocio, indicándolos como 100% compatibles

En este problema, se agrega un grado de complejidad al analizar la experiencia del usuario en los negocios, lo cual, en mi opinión, es una similitud más específica ya que al basarse en opiniones similares, se puede buscar opiniones positivas del usuario "similar" y recomendar dichos negocios a su par correspondiente, o filtrar opiniones negativas de la misma forma.

En resumen, en este problema se considera la similitud en base a las opiniones de los usuarios sobre los lugares que ambos visitaron, a diferencia del problema 2 donde sólo se identificaba similitud en base a los lugares visitados. Con respecto a los tiempos de ejecución, como se utilizó el mismo esquema y se cambió sólamente el cálculo del coeficiente, los tiempos de ejecución fueron bastante similares.

Conclusiones

A modo de concluir, la tarea fue muy útil para comprender el funcionamiento de la herramienta para Python MRJob, además de familiarizarme con el manejo de grandes volúmenes de datos.

Queda una sensación de derrota por el uso de diccionarios para resolver algunos problemas, ya que, me imagino que el uso de esta memoria en términos de espacio y tiempo puede perjudicar bastante el análisis para volúmenes más grandes, y a pesar de no tener éxito en las búsquedas online, mi idea es que debe existir una forma correcta de pasar los parámetros requeridos a través de los pasos Map/Reduce de MRJob, pero por motivos de tiempo y experiencia no se logró llegar a dicha solución.

Por otra parte, los tiempos de ejecución fueron bastante decentes (menores a un minuto) para los problemas 1 y 3. Mientras que para los problemas 2 y 4 llegó a tomar casi 10 minutos con los sets de 30000 datos, lo cual a mi parecer es bastante tiempo, considerando que dicho set presenta un 0.6% del tamaño del set de datos original.

Puede que el cálculo de similitud sea una operación que requiera de bastante poder de procesamiento y tiempo para realizarse, justificando los altos tiempos de ejecución, pero existe la sensación de que dichos resultados podrían haber sido mejores utilizando de mejor manera las herramientas de MRJob.
Pero en fin, se obtuvo los resultados esperados en un tiempo razonable, considerando que se trabajó con el tamaño de datos especificado (10000 y 30000 datos).

A pesar de esto, la tarea fue una gran fuente de aprendizaje y práctica, se logró analizar conjuntos de volumenes considerables de datos, encontrar estadísticas de interés y familiarizarse con el software y técnicas en cuestión.

A lo largo del desarrollo de la tarea, principalmente se investigó en el foro de StackOverflow y en la Documentación de MRJob.
Además, también se utilizó las siguientes fuentes para comprender mejor el funcionamiento de MRJob y la estructura de los pasos:

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