Trabajo de minería de texto: Análisis de reviews

Jaime García Lozano

30 de abril de 2022

1. Introducción

En este trabajo se buscará elaborar un método para ser capaces de extraer información relevante a partir de textos, concretamente reviews escritas por clientes de varios hoteles.

Se dispone de un catálogo de reviews de múltiples hoteles de diez ciudades diferentes: Pekín, Chicago, Dubai, Las Vegas, Londres, Montreal, Nueva Delhi, Nueva York, San Francisco y Shangai [1] . En el siguiente enlace es posible descargar todos los datos: [2].

La carpeta *hotels* será la que se utilizará. En ella, se puede encontrar una subcarpeta por ciudad (con el nombre de esta), que a su vez contiene ficheros con las reviews de cada hotel (cada uno también con el nombre de este).

Se plantean las siguientes preguntas:

1. **¿Qué partes de una habitación son las más mencionadas en cada hotel? P.ej. baño, cama, jacuzzi, ...**
2. **¿Qué servicios pueden detectarse por cada hotel? Por ejemplo, gimnasio, spa, piscina, restaurante, ...**
3. **¿Qué lugares y qué porcentaje de revisiones sobre cada ciudad mencionan otras ciudades, regiones o puntos de interés turístico? Por ejemplo, con vuestra información podríamos llegar a responder a la pregunta: ¿en Las Vegas se menciona el Gran Cañón del Colorado?**

Nota: Para acceder al código programado entrar el siguiente enlace [3].

2. Estructuración de los datos

En un inicio se planteó la posibilidad de crear un *json* en el que, como primeras claves, se tuviesen los nombres de las ciudades. Cada ciudad contendría un diccionario con los nombres de los hoteles como claves. Por último, la celda de cada hotel estaría ocupada por una lista con sus correspondientes reviews.

Es una estructura muy inmediata y de fácil acceso a la que se podría aplicar la sintaxis de consultas de *mongodb*. Sin embargo, a la hora de aplicar funciones, me veía obligado a generar bucles computacionalmente muy ineficientes. Se podría haber profundizado más en este aspecto, pero finalmente se optó por transformar el *json* en un *dataframe* (df) de la librería *pandas* de Python.

El df tiene dos niveles de índices: ciudad y hotel. Para cada índice se tiene celdas ocupadas cada una de ellas por una review. Aunque el df nos aporta muchas ventajas, tiene la limitación de que no puede haber filas de distintas longitudes. Por tanto, el número de columnas está condicionado por el hotel con más reviews. Las celdas vacías han de ser ocupadas por *NaNs*, lo que puede causar una carga de memoria bastante innecesaria. Aunque mejorable, en este caso de estudio la estructuración elegida no ha causado ninguna ralentización.

	beijing china_beijing_aloft_beijing_haidian	china_beijing_ascott_beijing
0	Oct 12 2009 \tNice trendy hotel location not t...	Nov 17 2009 \tgreat room layout service value-...
1	Sep 25 2009 \tGreat Budget Hotel!\tStayed two ...	Nov 7 2009 \tA quality hotel with quality serv...
2	Aug 4 2009 \tExcellent value - location not a ...	Oct 16 2009 \tclean rooms good location poor s...

Tabla 1: Estructura del df. Se muestran las dos primeras filas y dos primeras columnas. Nota: Se ha hecho la transpuesta para que cupiera mejor.

3. Idioma

A simple vista, parece que la mayoría de las reviews están escritas en inglés, pero nada nos garantiza que no haya un importante porcentaje en otros idiomas. De ser así, se debería pensar un método que fuera independiente del idioma. Por ejemplo, se podría desarrollar una función que detectara el idioma de cada celda y de no estar en inglés, la tradujera; lo cual supondría un enorme coste computacional, ya que las librerías de detección de idiomas y de traducción no son precisamente rápidas. Ante la sospecha de que el número de reviews en otros idiomas es bastante pequeño se ha hecho el siguiente análisis:

1. Creación de una función (*detect_language*) que tiene como *input* un texto y como *output* el idioma de este. Para ello se ha utilizado la librería *langdetect*.
2. Programación de un bucle que recorre todos los hoteles de cada ciudad. Para el conjunto de celdas de cada hotel, se eliminan las ocupadas por *NaNs* y se aplica la función *detect_language* al resto.
3. Finalmente tenemos otro df con la misma estructura que el original, en el que cada celda (ocupada previamente por una review) contiene el idioma de esta.

Con este nuevo df es sencillo calcular la proporción de reviews en inglés de cada hotel y mirar en qué hoteles es menor al 50 %:

		eng_prop	reviews_len
london	uk_england_london_andrews_house_hotel	0.47	16
	uk_england_london_new_pembury_central_park	0.45	10
	uk_england_london_viking_hotel	0.45	21
shanghai	china_shanghai_renaissance_shanghai_zhongshan_park_hotel	0.36	15

Tabla 2: Hoteles con una proporción de reviews en inglés menor al 50 %. Las columnas *eng_prop* y *reviews_len* son la fracción de reviews en inglés y el número de reviews respectivamente.

De todos los hoteles, solo hay 4 con menos de la mitad de las reviews en inglés. Podemos ver, además, que son porcentajes cercanos al 50 % y que tienen un número relativamente elevado de reviews. Por tanto, se ha decidido eliminar todas aquellas que estén en otros idiomas.

4. Partes de la habitación más mencionadas

En este apartado se ha optado por utilizar expresiones regulares (ER). La idea es tener un programa en el que el usuario pueda introducir las partes de la habitación de su

interés y que se devuelva el número de menciones de cada una de ellas para cada hotel.

A modo de ejemplo, se han utilizado las siguientes palabras: *room, bed, wardrobe, closet, sofa, chair, table, balcony, carpet, toilet, jacuzzi, bathtub, kitchen, fridge* y *freezer*.

Aparecen los siguientes retos:

- Cómo evitar que las mayúsculas o los signos de puntuación influyan en los resultados. Por ejemplo, cómo hacer que 'room', 'Room' o 'room;' sea guardado como la misma palabra.
- Conseguir que se detecten tanto plurales como singulares y se guarden como la misma palabra.

4.1. Expresión regular

Primeramente, construimos una ER que encuentre todos los singulares y plurales de una palabra dentro de un texto. Por ejemplo, para la palabra *bed* quedaría de la siguiente manera:

```
\bbed[s]*\b
```

Sin embargo, para palabras con plurales irregulares habrá que cambiar ligeramente la estructura. Por ejemplo, para *balcony*:

```
\bbalcon[y|ies]\b
```

Por otra parte, distintas partes del alojamiento puede contener la palabra 'room' (*bathroom*, *livingroom*, *bedroom*, etc). Por ello, añadimos una ER que encontrará todas las palabras que contengan 'room' o sea directamente *room*:

```
\w*room[s]*\b
```

Salvo estas dos últimas excepciones, el resto de palabras siguen una misma estructura. Programamos una función que como *input* tendrá la lista de palabras y como *output* la ER regular que las encontrará todas:

```
\w*room[s]*\b|\bbed[s]*\b|\bwardrobe[s]*\b|\bcloset[s]*\b|\bsofa[s]*\b|...etc
```

4.2. Función 'rooms'

En este paso construimos una función llamada *rooms* cuyos *inputs* serán un texto y una expresión regular, y como *output* un objeto de clase *Counter* (un diccionario con ciertas propiedades especiales) que contendrá la cantidad de menciones de cada palabra encontrada mediante la ER.

Seguirá el siguiente proceso:

1. Se quitan todas las mayúsculas, signos de puntuación y números del texto introducido.
2. Se le aplica ER.
3. Para cada elemento extraído en el paso anterior, se extrae el lema mediante la librería *nltk*. Con este paso hacemos que las palabras que estén en plural pasen a singular.
4. Finalmente guardamos el lema en un *Counter*, de la librería *collections*.

De esta manera, dispondremos de un nuevo df (*hotel_info*) con una columna (*room_info*) que contendrá el resultado obtenido para cada hotel.

		room_info
beijing	china_beijing_aloft_beijing_haidian	{'room': 8, 'bed': 4, 'fridge': 1, 'table': 1}
	china_beijing_ascott_beijing	{'room': 54, 'bedroom': 20, 'kitchen': 17, 'ba...
	china_beijing_autumn_garden_courtyard_hotel	{'room': 8, 'bathroom': 3, 'toilet': 1, 'subwa...
	china_beijing_bamboo_garden_hotel	{'room': 69, 'bed': 21, 'bathroom': 17, 'bedro...
	china_beijing_beijing_century_towers	{'room': 12, 'bedroom': 3, 'kitchen': 2, 'bed'...

Tabla 3: Estructura del df *hotel_info*. Se muestran las primeras filas.

Por ejemplo, si quisiéramos acceder a las partes de la habitación más citadas de un hotel de Londres llamado *Queensgate London Apartments*:

```
{'room': 24,
 'bedroom': 19,
 'bathroom': 12,
 'kitchen': 12,
 'bed': 10,
 'balcony': 6,
 'sofa': 6,
 'carpet': 3,
 'wardrobe': 2,
 'fridge': 2,
 'toilet': 2,
 'boothrooms': 1,
 'table': 1,
 'freezer': 1,
 'closet': 1}
```

5. Servicios

Para este apartado seguiremos el mismo proceso que en el anterior. A modo de ejemplo intentaremos detectar si los hoteles disponen de los siguientes servicios: *gym, swimming pool, restaurant, pub, disco* y *spa*.

Al df *hotel_info* se le añade la columna *services*:

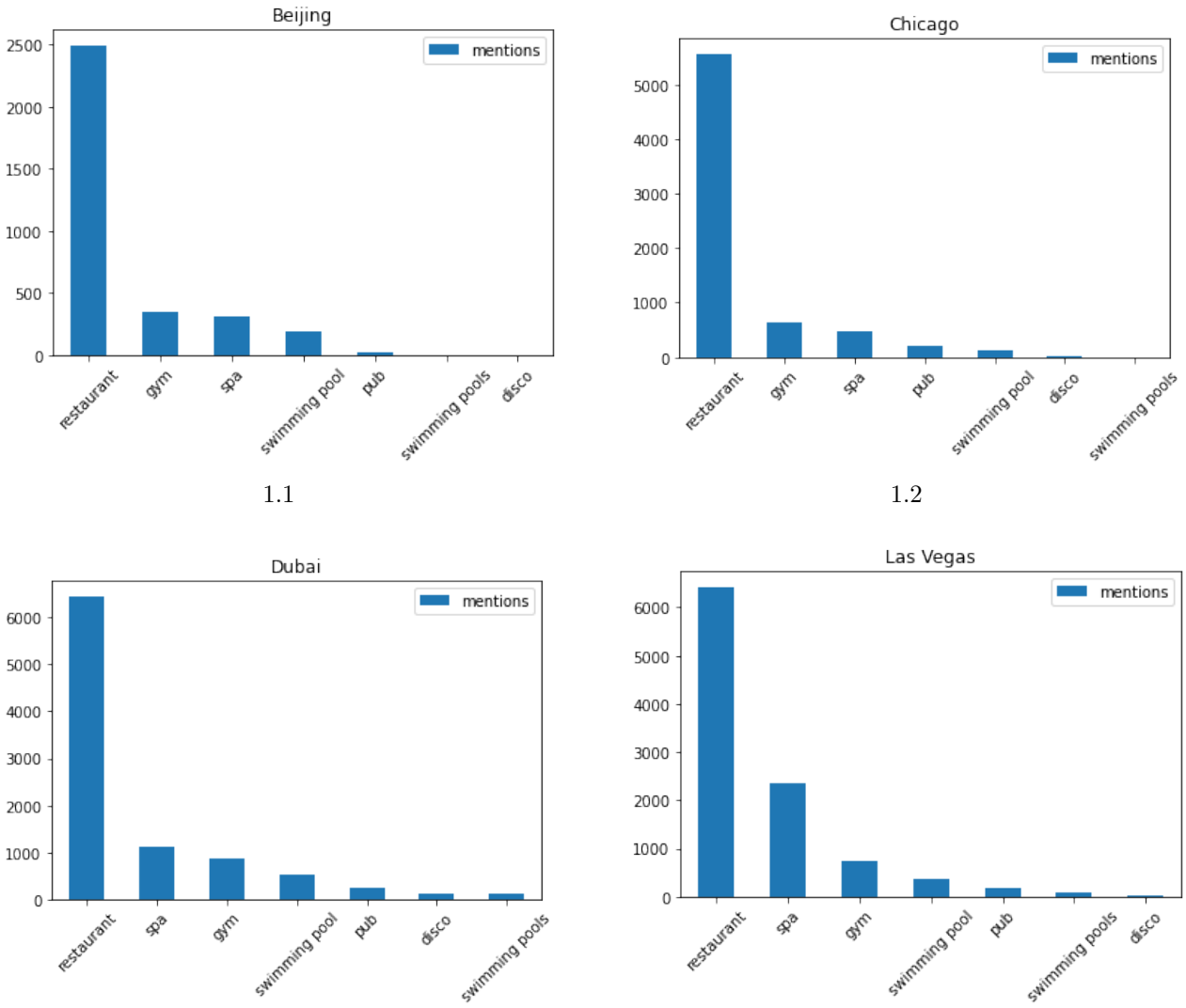
		services
beijing	china_beijing_aloft_beijing_haidian	{'gym': 2, 'restaurant': 2, 'swimming pool': 1}
	china_beijing_ascott_beijing	{'restaurant': 20, 'gym': 10, 'swimming pool': 1}
	china_beijing_autumn_garden_courtyard_hotel	{}
	china_beijing_bamboo_garden_hotel	{'restaurant': 28, 'spa': 1}
	china_beijing_beijing_century_towers	{'restaurant': 1}

Tabla 4: Estructura de la columna services. Se muestran las primeras filas.

Si estuviéramos interesados en saber qué servicios ofrece el hotel de Londres *Queensgate London Apartments*:

```
{'restaurant': 6, 'pub': 1}
```

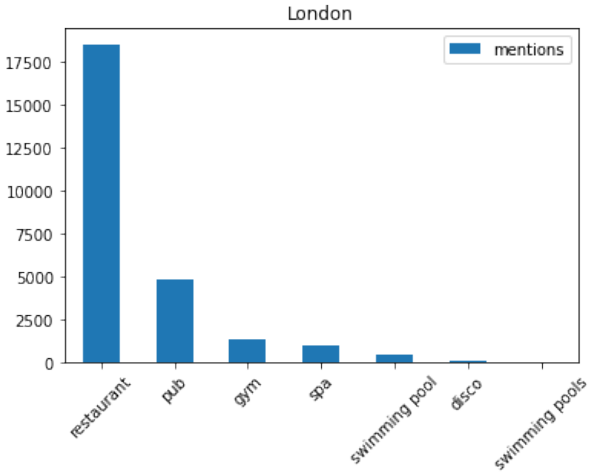
También sería interesante ver qué servicios son más comunes en cada ciudad:



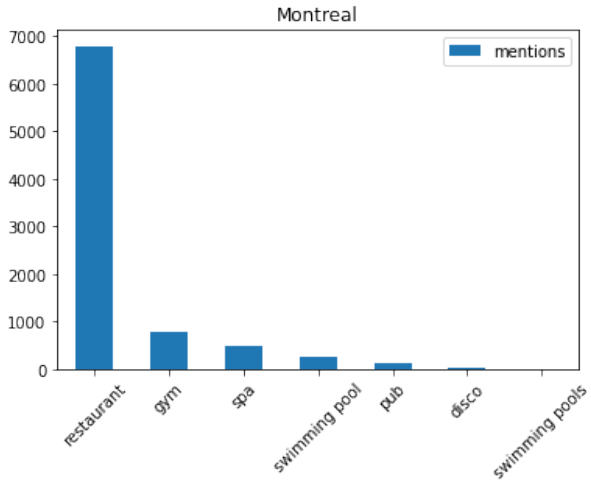
1.1 1.2

1.3 1.4

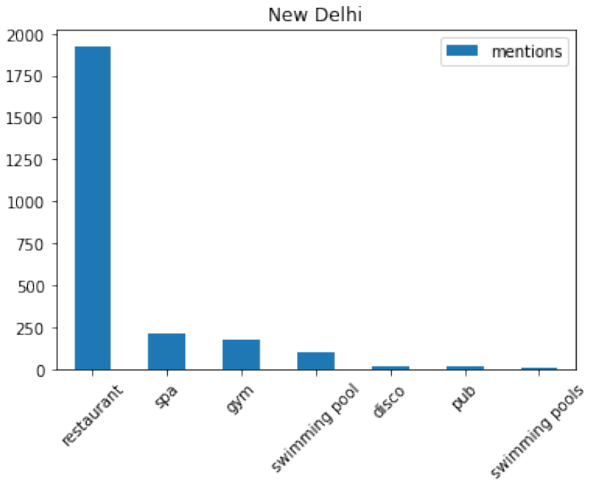
Figura 1: Gráfico de barras con la cantidad de menciones por servicio en cada ciudad. Primeras cuatro ciudades.



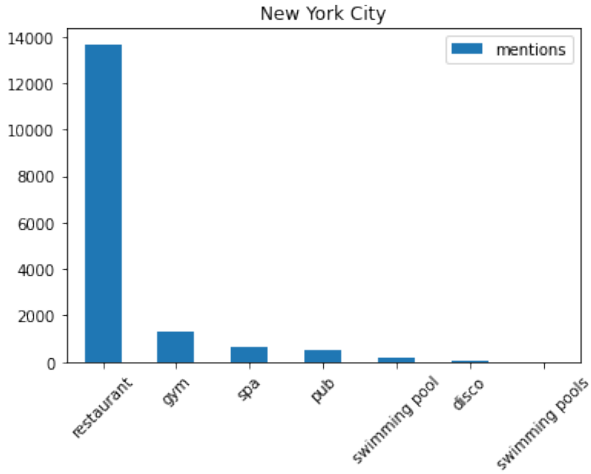
2.1



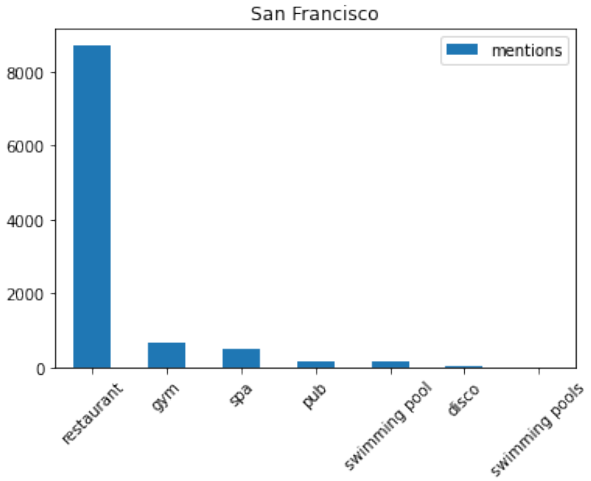
2.2



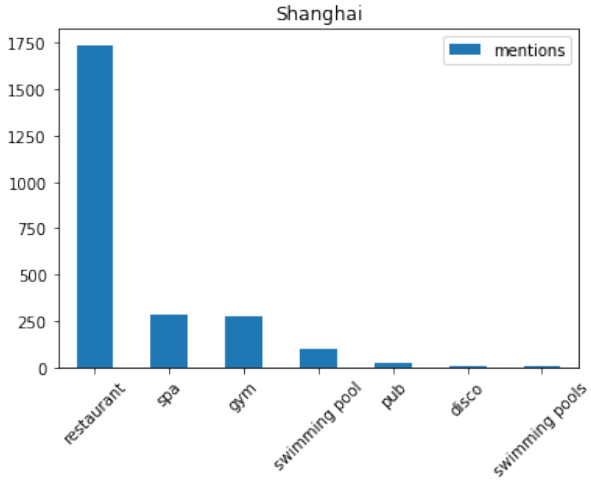
2.3



2.4



2.5



2.6

Figura 2: Gráfico de barras con la cantidad de menciones por servicio en cada ciudad. Seis ciudades restantes.

6. Lugares mencionados

En los dos anteriores apartados, las expresiones regulares han dado buenos resultados. Sin embargo ahora nos encontramos ante un reto en el que su aplicación de poco sirve: ¿Cómo encontramos nombres propios de ciudades, lugares geográficos, sitios turísticos, etc?

Una solución es el uso de entidades: la librería *Spacy* nos permite encontrarlas y etiquetarlas en un texto dado. Hay tres etiquetas que nos interesan:

- GPE: corresponde a entidades geopolíticas (ciudades, países, etc.).
- LOC: lugares geográficos como montañas, playas, etc.
- FAC: instalaciones (puertos, aeropuertos, ect.).

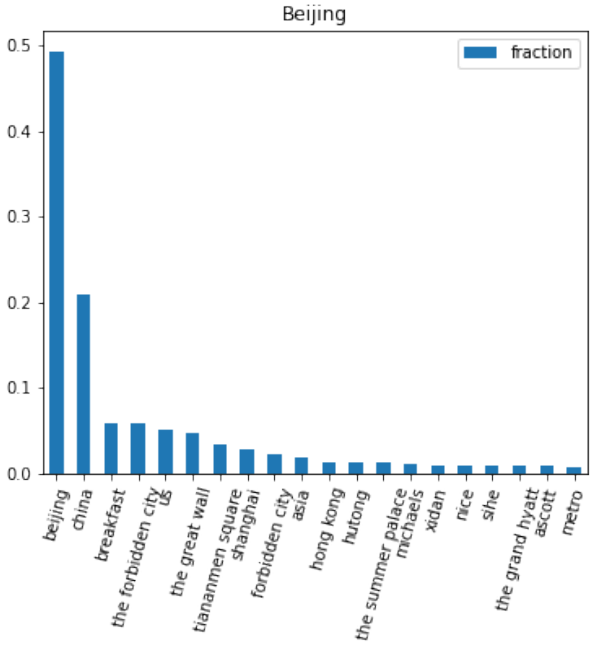
Definimos una función llamada *places_of_interest*, cuyo funcionamiento será el siguiente:

1. Tendrá como *input* un texto.
2. Tockeniza el texto.
3. Para cada entidad dentro del texto se filtran aquellas que tengan una de las tres etiquetas previamente mencionadas.
4. El *output* será un diccionario para cada review con los lugares mencionados.

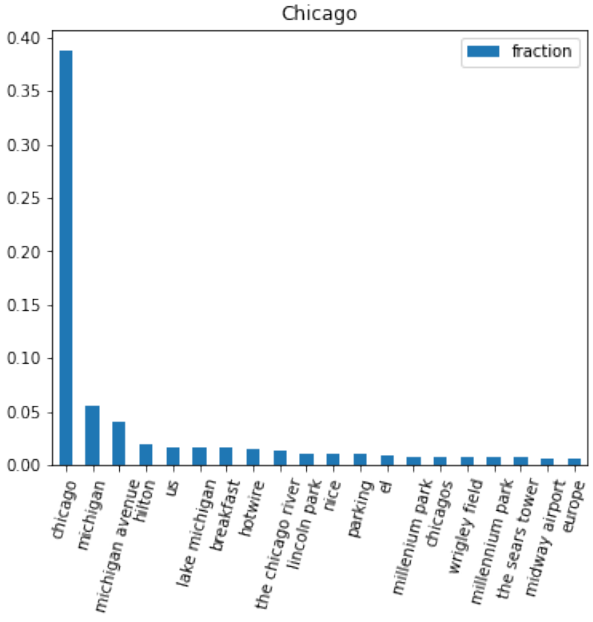
El problema de este apartado es que nos encontramos ante una situación computacionalmente muy ineficiente (la ejecución puede tardar en torno a 2 días).

Para acelerar el proceso se ha optado por escoger una muestra aleatoria de 50 hoteles para cada ciudad, y dentro de cada uno de estos hoteles, si el número de reviews es mayor a 50, también extraer 50 de manera aleatoria. Con esto se ha conseguido reducir el tiempo de ejecución a dos horas.

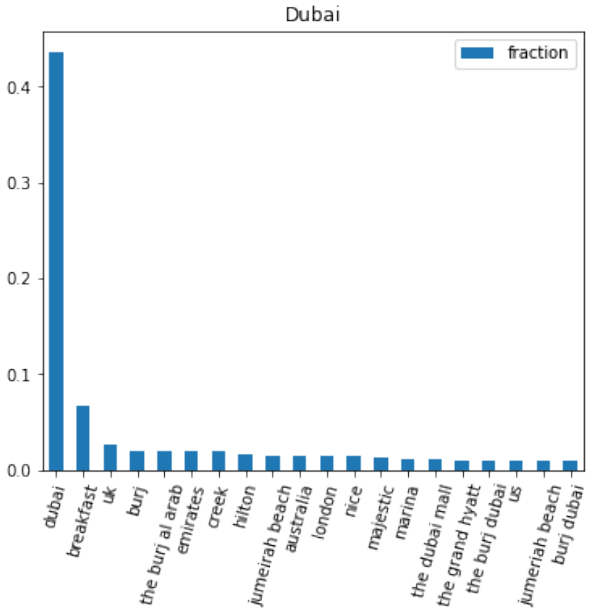
A continuación se muestran unos gráficos de barras con los diez lugares más mencionados en cada ciudad:



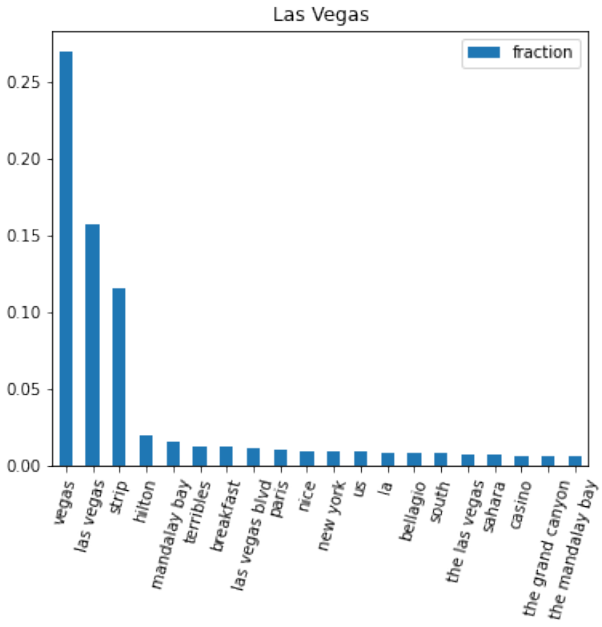
3.1



3.2

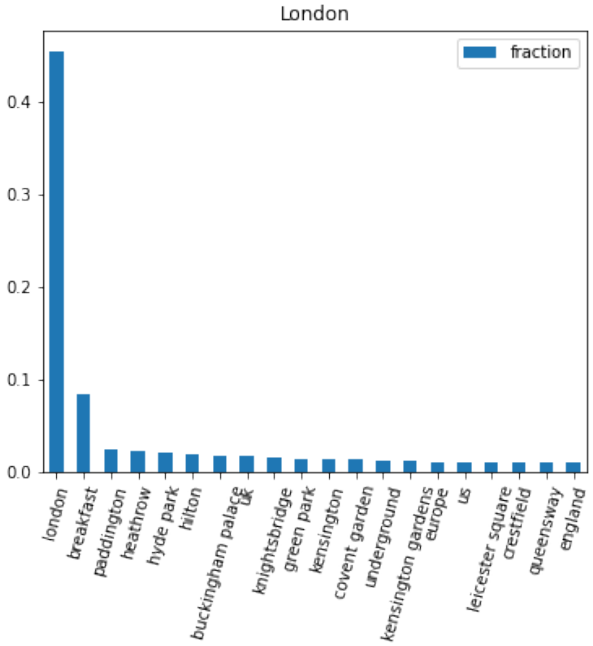


3.3

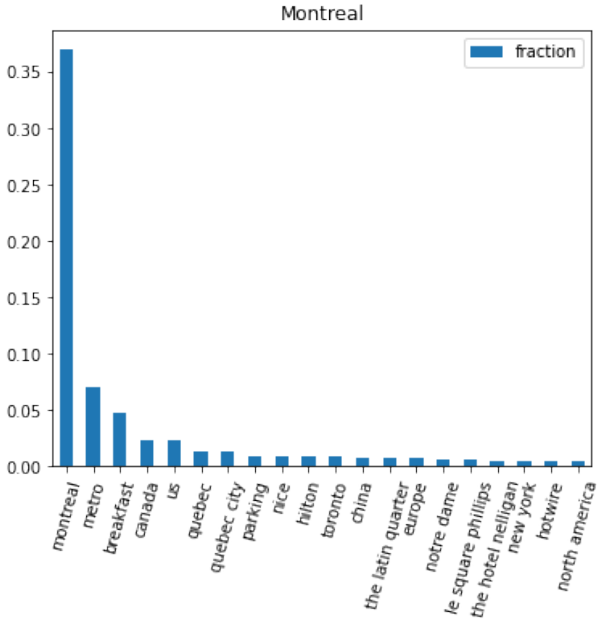


3.4

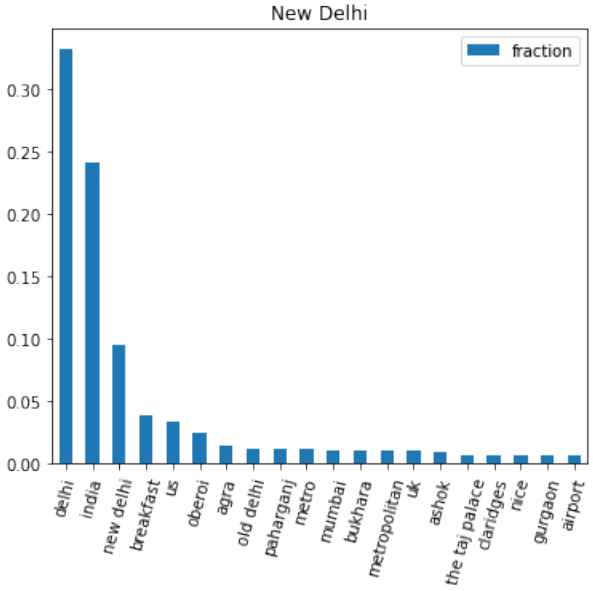
Figura 3: Gráfico de barras con la proporción de reviews que mencionan cada lugar. Primeras cuatro ciudades.



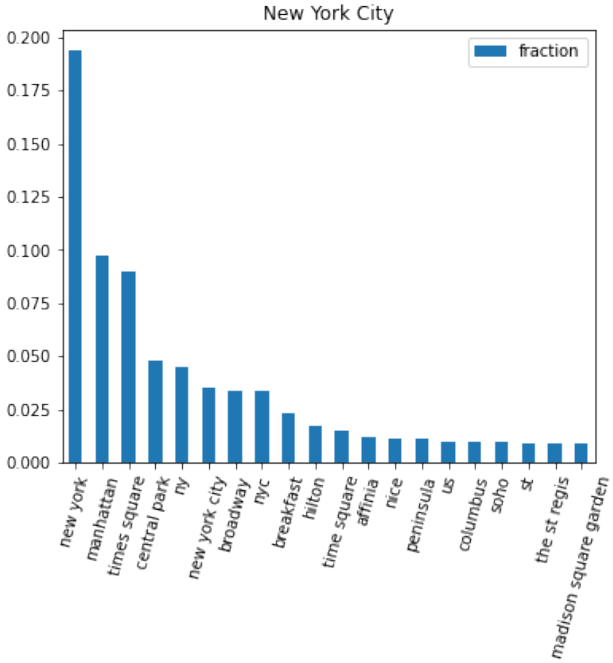
4.1



4.2

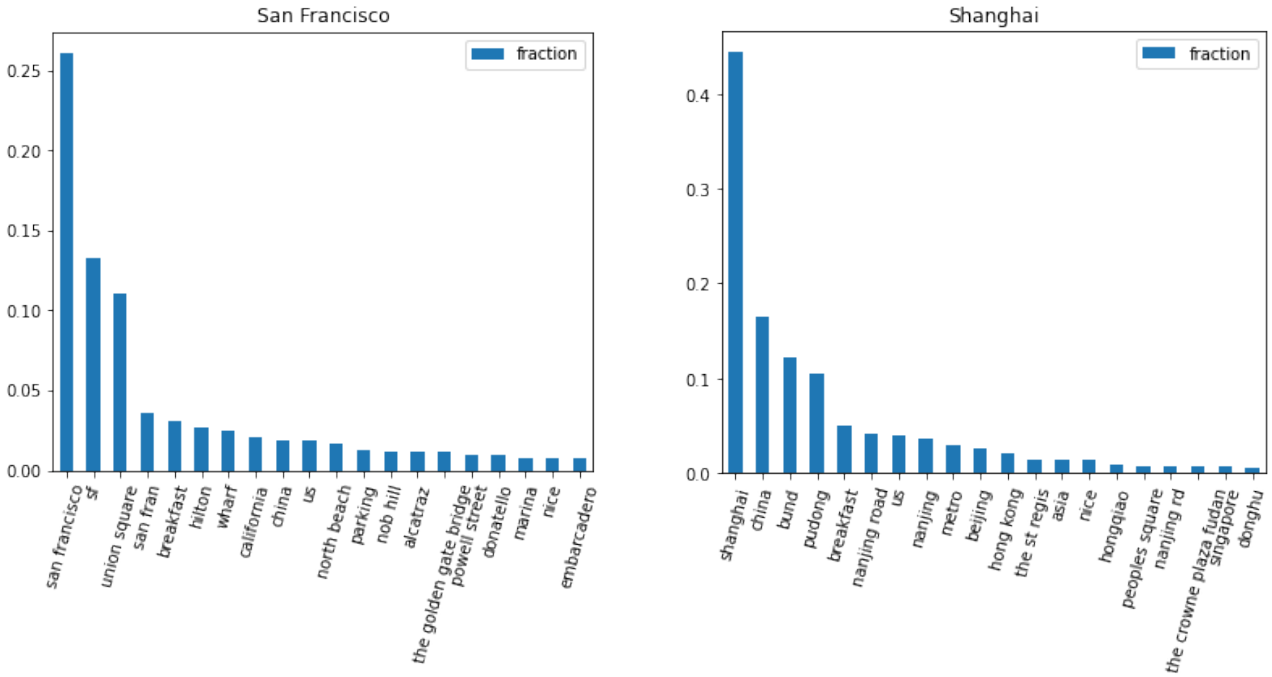


4.3



4.4

Figura 4: Gráfico de barras con la proporción de reviews que mencionan cada lugar. Cuatro ciudades siguientes.



5.1 5.2

Figura 5: Gráfico de barras con la proporción de reviews que mencionan cada lugar. Últimas dos ciudades.

7. Conclusión

En los dos primeros apartados se ha conseguido elaborar un código eficiente que consigue encontrar la inmensa mayoría de variaciones que pueden tener las palabras introducidas. En general, da buenos resultados, aunque se pueden ver algunos fallos como es el caso de *swimming pools*, que al estar formada por dos palabras el programa no consigue extraer su singular correctamente.

Otro punto mejorable sería la limpieza posterior de las palabras encontradas. Por ejemplo, está claro que en la mayoría de ocasiones *room* es utilizada como sustituta de *bedroom*, por lo que ambas tienen el mismo significado. Otro caso parecido sería el de *toilet* y *bathroom*.

En el último apartado la única limitación que se tiene es el inmenso coste computacional que supone tockenizar y extraer las entidades de cada review. Salvo eso, se han obtenido unos resultados convincentes (quitando *breakfast*).

Referencias

[1] K. Ganesan y C. Zhai, «Opinion-based entity ranking», Information retrieval **15**, 116-150 (2012).

[2] *Catálogo de reviews*, http://kavita-ganesan.com/entity-ranking-data/.

[3] *Repositorio*, https://github.com/JGL98/Analisis-reviews-Mineria-de-Texto.git.