AirBnB: Important Terms & Alternative Listings

Thomas Kleiven og Karsten Standnes

March 2017

1 Introduction

This assignment consists of analysing datasets containing information regarding listings at AirBnB. In order to analyse the data, we have used the Apache Spark framework a long side with python.

AirBnB is an online hospitality service, that allow people to list or rent short-term lodging like holiday rentals, aparment rentals, hotel rooms, etc. The company itself does not own any lodging, but receives commissions from both guests and hosts in conjunction with each booking. It has more than 3.000.000 lodgning listings in 65.000 cities in 191 countries. The dataset we have been given focuses on three cities, namly Seattle, San Fransisco and New York. Furthermore, the data sets contains information about each listing such as when it was/have been available for rental, customer reviews, etc.

Figure 1: Spark logo

Contents

2 Important Terms

2.1 Running the program

In order to get the 100 most important words according to their weight, the program has to be run like

python tf_idf.py /home/tkleiven/Documents/Skole/BigData/task2/ -n "Williamsburg"

or

python tf_idf.py /home/tkleiven/Documents/Skole/BigData/task2/ -l "42"

You will then get a .TSV-file in the same folder as your input path, but note that all datasets you are using has to be present in the given input path. In addition, the program is by default looking for 'listings_ids_with_neighborhoods.tsv' and 'listings_us.csv'. Enjoy.

2.2 General information

The weight of a word is given by

$$weight_{t,d} = tf_{t,d} \cdot idf_{t,d} \tag{1}$$

$$tf_{t,d} = \frac{Number\ of\ times\ term\ t\ appears\ in\ a\ document\ d}{Total\ number\ of\ terms\ in\ the\ document\ d} \tag{2}$$

$$idf_{t,d} = \frac{Total\ number\ of\ documents}{Number\ of\ documents\ with\ term\ t\ in\ it} \tag{3}$$

Where document d in this scenario is the listing's description.

First, we make a dataframe out of our listings_us.csv dataset and the dataset provided with the neighborhood name and ID. Secondly, we join these two dataframes like we did in the previous assignment in order to get the description of either a listingID or a neighborhood. For clarification, we define the term 'documents' in the following ways

1. When ID is given as input, we define a document as the description of a listing. The number of documents will then be the number of different descriptions.

2. When a neighborhood is given as input, we define a document as all the descriptions within this neighborhood which form a neighborhood description. The number of documents will then be the number of neighborhood descriptions. For example, all the descriptions in the neighborhood Williamsburg is now one document, while all the descriptions in East Harlem is also one document.

2.3 Calculation of $tf_{t,d}$

In order to compute the $tf_{t,d}$ we have to count the number of times a term t appears in a document d. The algorithm works in the following way

1. We make an RDD on the form $< key, value >$ where ID or neighborhood works as the key, and the description of the listing works as the value. Depending on the input, the value can take both one description for a given listing ID, or all the descriptions for each listing within a neighborhood.

2. Let us use the filter function on the RDD in order to obtain only the listing or neighborhood given as input.

3. We then clean the description by applying lowercase and removing special characters.
4. Let us apply the flatMap function and split on spaces in order to get our description into a collection of words.
5. The number of terms in the document is now easily calculated by applying the distinct and the count functions.
6. The number of times a term t appears in a document d is now given by applying the map function to get each word in the description on the form $(word, 1)$. Lastly, we can count the number of times each word appears in the document by applying

$$.reduceByKey(lambda\ a, b : a + b) \tag{4}$$

The reader is encouraged to have a look at the provided code to get more details about the algorithm.

2.4 Calculation of $idf_{t,d}$

In order to compute the $idf_{t,d}$ we have to count the total number of documents and the total number of documents containing a word w. The algorithm works in the following way

1. The total number of documents will be as defined in Section 2.
2. As in the calculation of the $tf_{t,d}$, let us use the filter function on the RDD in order to obtain only the listing or neighborhood given as input.
3. We remove special characters from the descriptions and we split each document on spaces. This is done by using the map and replace functions on the RDD. We then keep all the descriptions in a nested array, i.e. a two dimensional array.
4. Let us iterate over all the documents in order to calculate the number of documents that contain a word w.
5. Voilà! We now have both the number of documents and the number of documents that contain a certain word w. The results are kept in a dictionary with each word as a key and the $idf_{t,d}$ as value.

The reader is encouraged to have a look at the provided code to get more details about the algorithm.

2.5 Calculate the weight of a word

The weight of the word is given by Equation 1. For easy accessing we have decided to keep our $tf_{t,d}$ and $idf_{t,d}$ two dictionaries, where each word is a key in the dict and the value is either the term-frequency or the inverse-term-frequency. We are now able to iterate over the words in the dict and multiply the $tf_{t,d}$ and $idf_{t,d}$ to get the desired result. In the end we select the top 100 words with the highest weight and write these words toghether with their weight to a TSV-file.

3 Alternative listings

3.1 Running the program

In order to get the top n alternative listings, the program has to be runned like

Alternative_listings.py <listing_id> <date:YYYY-MM-DD> <x> <y> <n> <folderpath>

The different inputs here in the walkthrough of the code under, there is also given a specific example is given under visualization 3.5

3.2 Requirements

In the second task we produce a list of alternative Airbnb lodgings based on a specific listing and input from the user. The specifications given was that the alternative listings:

- are vacant on the date,
- are the same roomtype
- 's price is not higher by more than x% than the original price
- are within y kilometers of the original geographical position

3.3 Formulas

In order to obtain the n best alternatives the formulas under are used:

1. The maximum price a alternatice listing can have is given by

$$price_{max} = price_{listing} \cdot \frac{100 + x}{100} \tag{5}$$

   which give a result in the same unit as *price_listing*.

2. The Haversine formula for calculating the distance between two listing values is given by

$$distance = 2 \cdot r \arcsin \sqrt{\sin^2(\frac{lat_2 - lat_1}{2}) + \cos(lat_1) \cdot \cos(lat_2) \cdot \sin^2(\frac{lon_2 - lon_1}{2})} \tag{6}$$

   where r is the radius of the earth which is 6371 km and lat, lon is short for latitude and longitude.

3. To find common amenities for two and two, a built-in intesect formula from the python library Numpy bee used. It has the form

$$common\_amenities\_count = \text{numpy.intersect1d}(common\_amenities_1, common\_amenities_2) \tag{7}$$

3.4 Finding the n best alternatives

In the task we find alternatives for a specific listing. To get the n best we have to calculate the maximum price an alternative can have, the distance from a listing to the selected listing and find the amount of common amenities. The results are then sorted based on these. More detailed, the code works in the following way

1. The user writes 6 variables to the program on the form. These are *id*, *date*, *x*, *y*, *n* and *folderpath*. In the same order these represent the id of the listing we find alternatives for, which date it must be available, maximum how much more it can cost in %, how far it can be from the selected listing, how many alternatives will be returned and the path to where the listing_us.csv and calender_us.csv datasets are located.

2. We do as in the previous task and make a dataframe from listings_us.csv and then a dataframe from calender_us.csv. Then we use the spark join function on *id* from the listings_us.csv- with *listings_id* from calender_us.csv dataframe.

3. The colums that are useful for the task is selected to give a more clean looking dataframe and minimizing the size of the output file.

4. The user input is processed. The program is taking the user given id to select a listing. To make it easier to access the attributes of this listing, an array of the listing is made.

5. Based on the input from the user and information gained from the selected listing, we can filter out the potensial listings. First are all the listings that are from another city are removed. Then we look at the maximum price the user is willing to pay using formula 5 and use the filter function to keep only the listings at or below this cost. Another demand was that the alternatives should be of the same room type, so all listings with a different room type are removed using filter. After this we use the Haversine formula - formula 6 - to calculate the the distance the listings are from the listing the user have chosen. After obtaining this value, we can filter out the listings within the max distance from the original listing.

6. We also need to know if the listings are available for renting on the date given. To be abel to do this the sorted listings dataframe is joined with the calender dataframe. This gives us the columns *data* and availability. Then the code return the listings that are available on the given date with the use of filter.

7. Then we use the function in formula 7 and find the amount of common amenities for each of the listings we have after filtering.

8. The results are sorted by number of common amenities descending. Up to n listings are written to file. The result file is named "alternativeListTopN.tsv" and is located in the same folder as the python code. As requested in the task it has the form
id \t name \t number_of_common_amenities \t distance \t price.

3.5 Visualization

We have chosen to visualize all the listings that are potential alternativives when calling

Alternativ_listings.py 8017041 2016-12-15 10 2 20 <folderpath>

. This is done by using the function "visualizationWriter" which will not in the version of the code delivered, but is provided to show the method. Under is a link to a Carto where all the listings are plotted. A cloropleth map is made based on the alternatives price. When a point is click a legend will show a picture from the listing, when it was last reviewed, the price and the review score rating.
Link: https://thomklei.carto.com/viz/d8291ad4-1539-11e7-b1fa-0ef24382571b/public_map

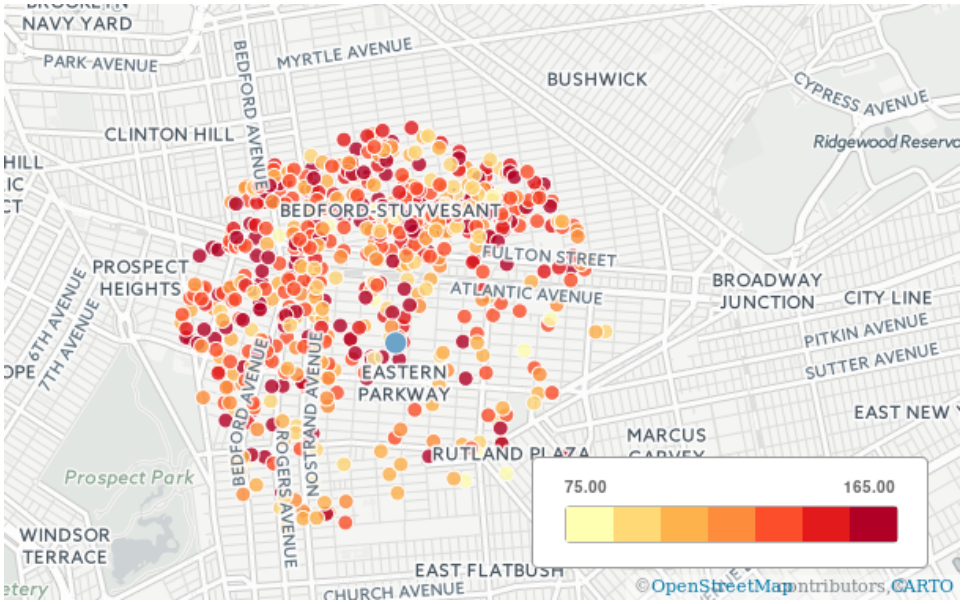

Figure 2: Visualization with carto, price rangeing from 75 to 165 $