Geographical rec rates

What's this about?

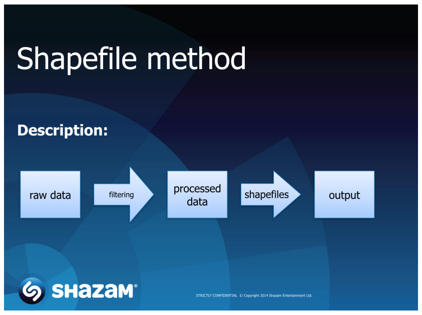
Nowadays, Shazam can only know the recognition rate for each country based on the country feature for each tag. What we are more interested in is the different performance at grained level, like which types of venues have better/worse recognition rate or which venue with lots of tags but perform not ideal enough. As for this project, I will explore some recognition rate patterns for points of interest on some certain area and pay more attention to those places with large number of tags and rather low recognition rate. I hope this project can help music team to identify bad-performed places correctly and improve our identification algorithm through some investigation on bad area. At the same time, we can also give a report about the stats of recognition rate weekly or monthly to give a better understanding of the updated performance of our algorithm.

Introduction to tools used:

- Shapefile: The Esri shapefile, or simply a shapefile, is a popular geospatial vector data format for geographic information system software. I use the polygon shapefiles to cover the outline of each building in order to calculate the recognition rate for venues. For details: Shapefile.wiki
- QGIS: QGIS is a cross-platform free and open source desktop geographic information system (GIS) application that provides data viewing, editing, and analysis capabilities. I use the powerful tool to view shapefiles with different layers of matched/unmatched data which give me the idea of cons and pros of some certain method and some pictures I show are just screenshots from the output of QGIS. For details: QGIS.wiki
- Bounding box: The online website I used to get the latitude and longitude border for the area I am interested in. It basically draw a rectangle that covers the area and then output the range of the latitude and longitude corresponding to the box. The website is here: boundingbox
- Google Places API: I use the Google Places API/Place Searches/Nearby Search Requests to help me output the places around centers with latitude and longitude by supplying the keywords and specifying type of place I am searching for. For the parameters and details: google_places_API google_API_alternative.py findtypes.py

Two main methods:

1. **Shapefile method**: Assign each point to polygon shapefiles directly to calculate the recognition rate
2. **Kmeans clustering method**: Use Clustering analysis to group the data into clusters according to latitude and longitude, then figure out the recognition rate for each cluster


Shapefile method

Description:

raw data
filtering
processed data
shapefiles
output
SHAZAM
STRICTLY CONFIDENTIAL © Copyright 2014 Shazam Entertainment Ltd.

As for this method, I use the raw data got from streaming tags and filter out the data with no geographical information and output each eligible tags with latitude, longitude and match status three features. With these information, We can assign each tags into certain polygons according to their latitude and longitude and then calculate the recognition rate simply by:

$$Rec\_rate = N\_match / (N\_match + N\_unmatch)$$

N_match: number of matched tags in the polygon

N_unmatch: number of unmatched tags in the polygon

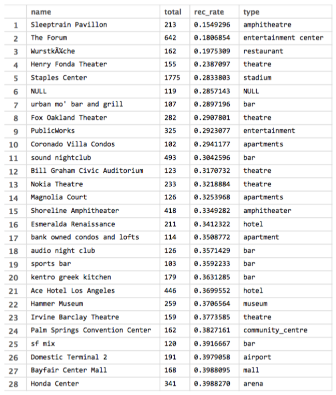
We can make it visualized by the following plots (take the union square in San Francisco for example) :

Original google street plot   plot with polygon shapefile on it   plot with both shapefile and tags on it

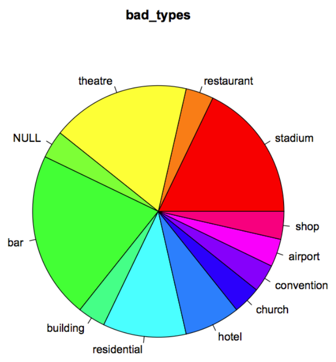
In this way, we can directly count the number of unmatched tags and matched tags in each building and give the recognition rate accurately and simply for each venue. For this method, I use the one month data in California State for an example, due to the difficulty to run the points in polygon function locally, I just got the result from the function run on cloud and group, process and clean the output, the plots shown below are the results I got based on California area, the related code is here:onemonthdata.R

The left shows the places with number of tags more than 100, recognition rate less than 0.4 sorted by recognition rate in ascending order. The right part is the pie chart corresponding to the types shown on the left.

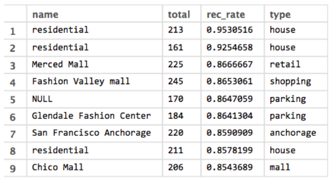
bad performed places   **pie chart for the bad performed types**



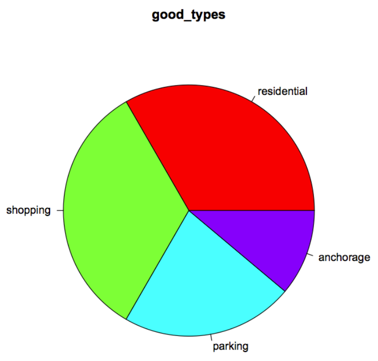
	name	total	rec_rate	type
1	Sleeptrain Pavillon	213	0.1549296	amphitheatre
2	The Forum	642	0.1806854	entertainment center
3	WurstkÃ¼che	162	0.1975309	restaurant
4	Henry Fonda Theater	155	0.2387097	theatre
5	Staples Center	1775	0.2833803	stadium
6	NULL	119	0.2857143	NULL
7	urban mo' bar and grill	107	0.2897196	bar
8	Fox Oakland Theater	282	0.2907801	theatre
9	PublicWorks	325	0.2923077	entertainment
10	Coronado Villa Condos	102	0.2941177	apartments
11	sound nightclub	493	0.3042596	bar
12	Bill Graham Civic Auditorium	123	0.3170732	theatre
13	Nokia Theatre	233	0.3218884	theatre
14	Magnolia Court	126	0.3253968	apartments
15	Shoreline Amphitheater	418	0.3349282	amphitheater
16	Esmeralda Renaissance	211	0.3412322	hotel
17	bank owned condos and lofts	114	0.3508772	apartment
18	audio night club	126	0.3571429	bar
19	sports bar	103	0.3592233	bar
20	kentro greek kitchen	179	0.3631285	bar
21	Ace Hotel Los Angeles	446	0.3699552	hotel
22	Hammer Museum	259	0.3706564	museum
23	Irvine Barclay Theatre	159	0.3773585	theatre
24	Palm Springs Convention Center	162	0.3827161	community_centre
25	sf mix	120	0.3916667	bar
26	Domestic Terminal 2	191	0.3979058	airport
27	Bayfair Center Mall	168	0.3988095	mall
28	Honda Center	341	0.3988270	arena



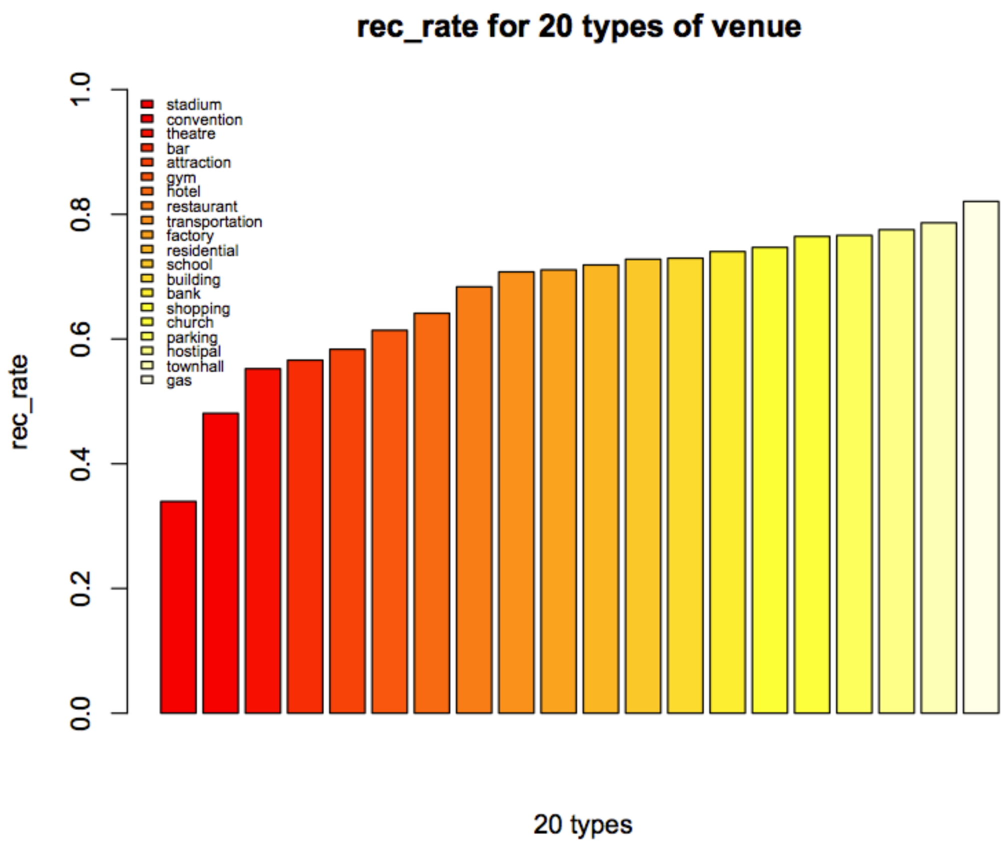
This time, the left side shows the good performed places with number of tags more than 150, recognition rate more than 0.85 sorted by recognition rate in descending order. The right part is the corresponding pie chart to the types shown on the left side.



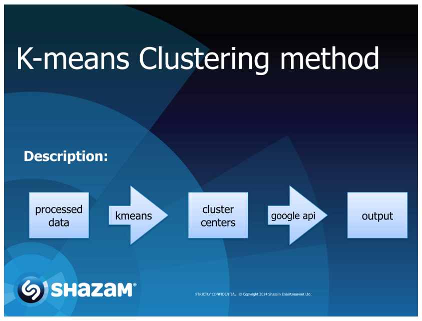
	name	total	rec_rate	type
1	residential	213	0.9530516	house
2	residential	161	0.9254658	house
3	Merced Mall	225	0.8666667	retail
4	Fashion Valley mall	245	0.8653061	shopping
5	NULL	170	0.8647059	parking
6	Glendale Fashion Center	184	0.8641304	parking
7	San Francisco Anchorage	220	0.8590909	anchorage
8	residential	211	0.8578199	house
9	Chico Mall	206	0.8543689	mall



With best and worst several places shown above, we can already get some information about the different performance for different types of venues. Stadium, bar and theatre occupy the first three places in the bad_types chart while shopping, residential and parking are among the first three places in the good_types chart. Then according to the 200+ different types given by the shapefile, I manually group them into 20 types according to the similarity and importance , the output below shows the recognition rate for each type:



We can tell from the plot that stadium, convention, theatre, bar and attraction are the 5 worst types of venues which agree with our common sense and best several places are just among the types like gas station, townhall, hospital which we are not that interested in.



K-means Clustering method

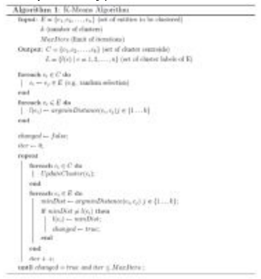
Description:

processed data → kmeans → cluster centers → google api → output

SHAZAM

STRICTLY CONFIDENTIAL © Copyright 2014 Shazam Entertainment Ltd.

As for this method, I use the processed data with the three key features: latitude, longitude, match_status in it. I perform the Kmeans clustering method based on the first two features latitude, longitude and then output the centers, the size for unmatched tags, the size for matched tags, the total number of tags for each cluster (we can also get the points belong to each cluster optionally). The kmeans code is here:kmeans.r and the basic Kmeans pseudocode is shown below:



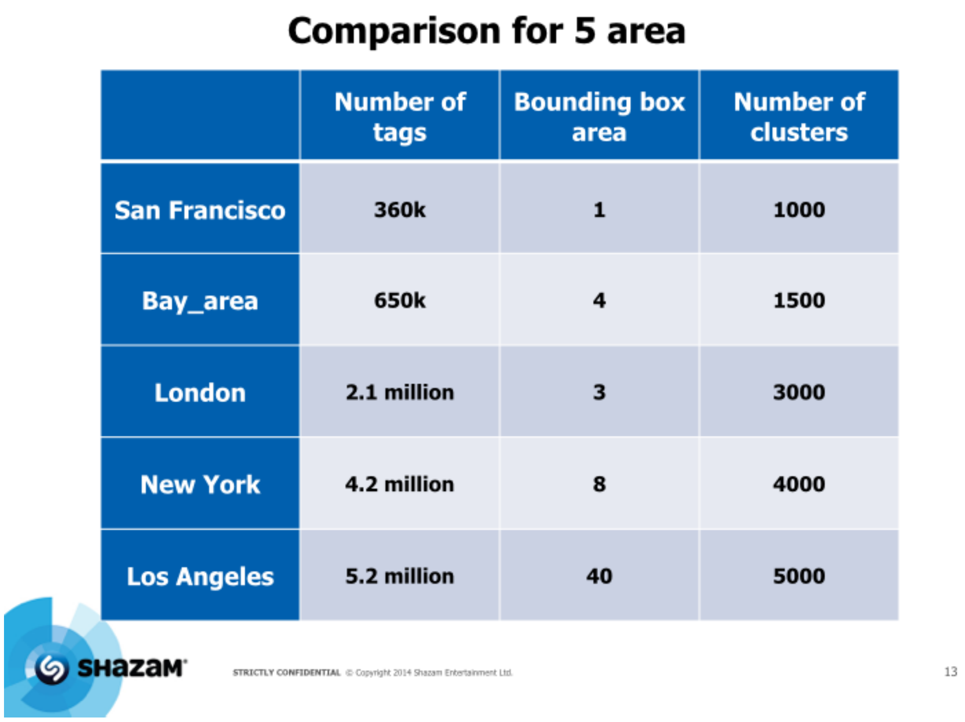
```
Algorithm 1: K-Means Algorithm
Input: $E = \{e_1, e_2, \ldots, e_n\}$ (set of entities to be clustered)
       $k$ (number of clusters)
       $MaxIters$ (limit of iterations)
Output: $C = \{c_1, c_2, \ldots, c_k\}$ (set of cluster centroids)
        $L = \{l(e) \mid e = 1, 2, \ldots, n\}$ (set of cluster labels of E)

foreach $c_i \in C$ do
    $c_i \leftarrow e_j \in E$ (e.g. random selection)
end
foreach $e_i \in E$ do
    $l(e_i) \leftarrow argminDistance(e_i, c_j) j \in \{1 \ldots k\}$
end

$changed \leftarrow false;$
$iter \leftarrow 0;$
repeat
    foreach $c_i \in C$ do
        $UpdateCluster(c_i);$
    end
    foreach $e_i \in E$ do
        $minDist \leftarrow argminDistance(e_i, c_j) j \in \{1 \ldots k\};$
        if $minDist \neq l(e_i)$ then
            $l(e_i) \leftarrow minDist;$
            $changed \leftarrow true;$
        end
    end
    $iter$++;
until $changed = true$ and $iter \leq MaxIters$;
```

With the information output by the kmeans clustering, we can either use the centers corresponding to bad recognition rate to use google places api automatically give us the venues around the points or manually find the bad-performed places on our own on the map. As for this case, I introduce the way and result given by :

This time I choose the five different area that we are most interested in, these places are either big cities with huge market or places near our office which makes us easy to tell the correctness or go on investigation. The table below shows the comparison for the 5 area we choose:



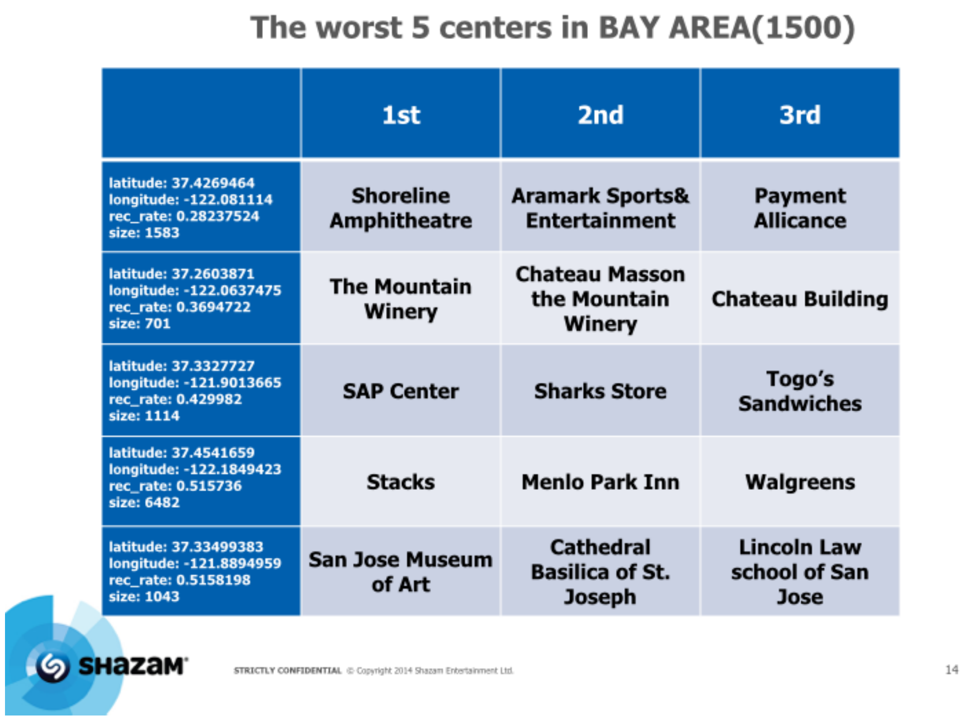
Comparison for 5 area

	Number of tags	Bounding box area	Number of clusters
San Francisco	360k	1	1000
Bay_area	650k	4	1500
London	2.1 million	3	3000
New York	4.2 million	8	4000
Los Angeles	5.2 million	40	5000

STRICTLY CONFIDENTIAL © Copyright 2014 Shazam Entertainment Ltd. 13

We choose the number of clusters based on the the number of tags, the bounding box area we choose and common sense. It's very hard to choose a perfect number of clusters for each area, since it's very time-consuming to run the process for one single decent number,if we plan to run the process for different numbers which are supposed to big and time-consuming each time, it's not applicable in that case. What's more, even if with a best value as the number of clusters for some criterion, we are not sure the number will be the one suitable for our identification case. So we just choose several numbers to tune the parameter for a place by some common sense and then make it approximately scalable for other places. In this way, I give the number of clusters for these 5 cities.

With the position of the centers for each cluster after the Kmeans clustering analysis, I use the google API to return to the first 3 (if there are more than 3 results) results for each center according to keyword and parameter I set for the API. As for the results I shown below, I use rankby = prominence and the radius = 200. The keyword and parameter I use can be tuned to be better for different situations which I will talk about it later.

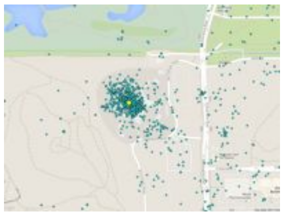


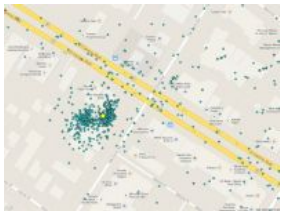
The worst 5 centers in BAY AREA(1500)

	1st	2nd	3rd
latitude: 37.4269464 longitude: -122.081114 rec_rate: 0.28237524 size: 1583	Shoreline Amphitheatre	Aramark Sports& Entertainment	Payment Allicance
latitude: 37.2603871 longitude: -122.0637475 rec_rate: 0.3694722 size: 701	The Mountain Winery	Chateau Masson the Mountain Winery	Chateau Building
latitude: 37.3327727 longitude: -121.9013665 rec_rate: 0.429982 size: 1114	SAP Center	Sharks Store	Togo's Sandwiches
latitude: 37.4541659 longitude: -122.1849423 rec_rate: 0.515736 size: 6482	Stacks	Menlo Park Inn	Walgreens
latitude: 37.33499383 longitude: -121.8894959 rec_rate: 0.5158198 size: 1043	San Jose Museum of Art	Cathedral Basilica of St. Joseph	Lincoln Law school of San Jose

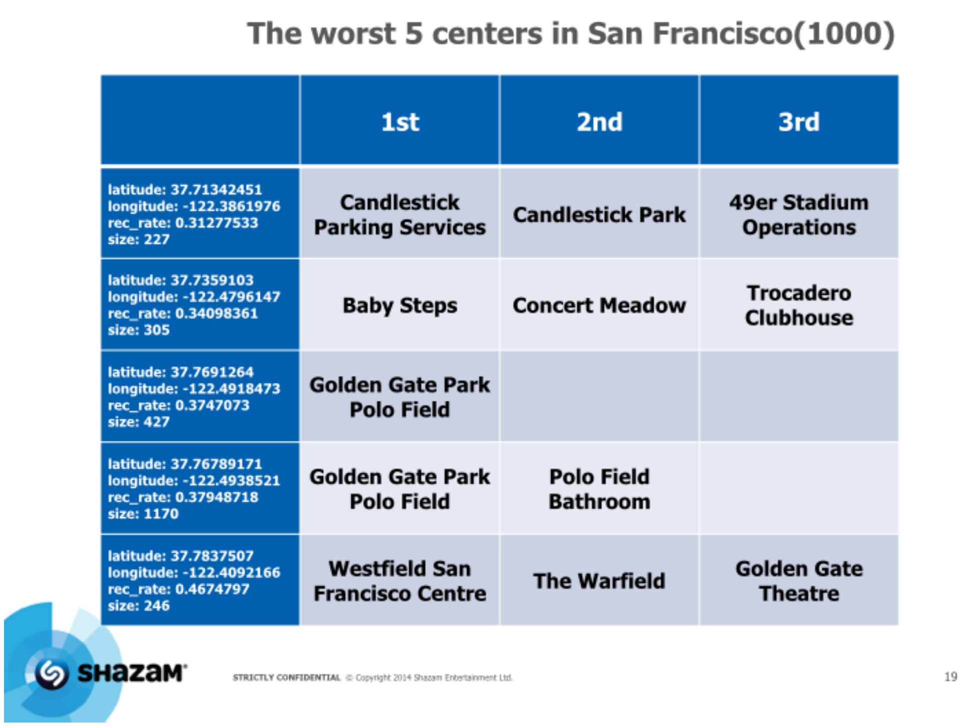
STRICTLY CONFIDENTIAL © Copyright 2014 Shazam Entertainment Ltd. 14

Example screenshots:





shoreline amphitheatre area Shazam office areain Menlo Park (the 4th row shown on the column)
green points are tags around that area, yellow point is the center we found green points are tags and yellow one is the center

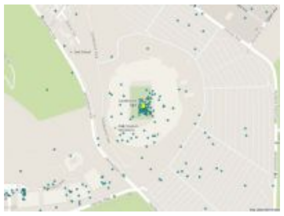


The worst 5 centers in San Francisco(1000)

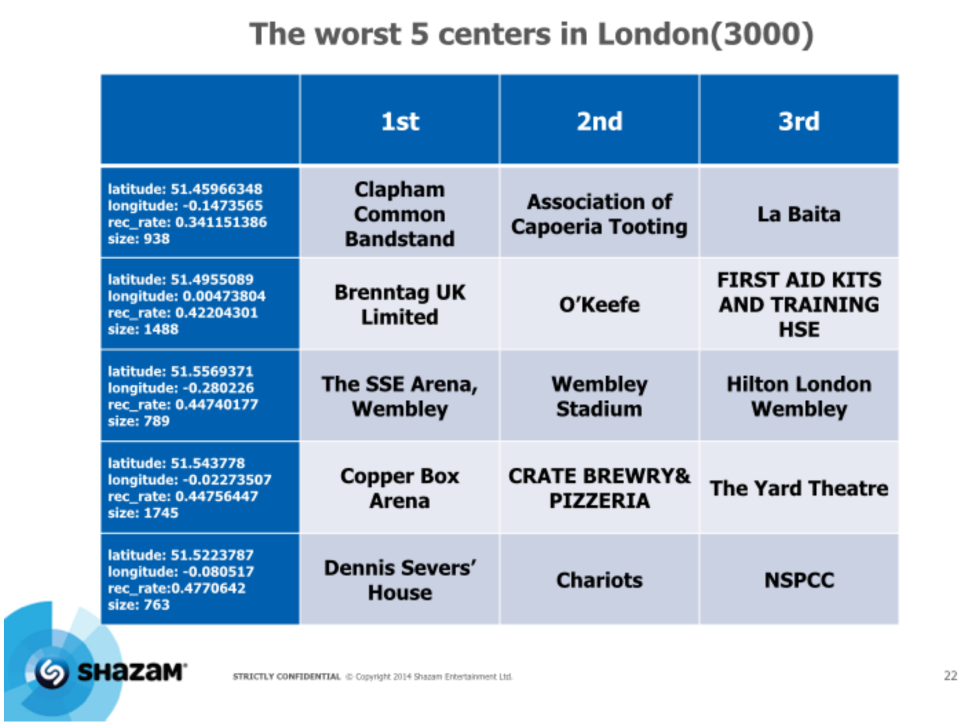
	1st	2nd	3rd
latitude: 37.71342451 longitude: -122.3861976 rec_rate: 0.31277533 size: 227	Candlestick Parking Services	Candlestick Park	49er Stadium Operations
latitude: 37.7359103 longitude: -122.4796147 rec_rate: 0.34098361 size: 305	Baby Steps	Concert Meadow	Trocadero Clubhouse
latitude: 37.7691264 longitude: -122.4918473 rec_rate: 0.3747073 size: 427	Golden Gate Park Polo Field		
latitude: 37.76789171 longitude: -122.4938521 rec_rate: 0.37948718 size: 1170	Golden Gate Park Polo Field	Polo Field Bathroom	
latitude: 37.7837507 longitude: -122.4092166 rec_rate: 0.4674797 size: 246	Westfield San Francisco Centre	The Warfield	Golden Gate Theatre

STRICTLY CONFIDENTIAL © Copyright 2014 Shazam Entertainment Ltd. 19

Example screenshot:



Candlestick Park area(Stadium)



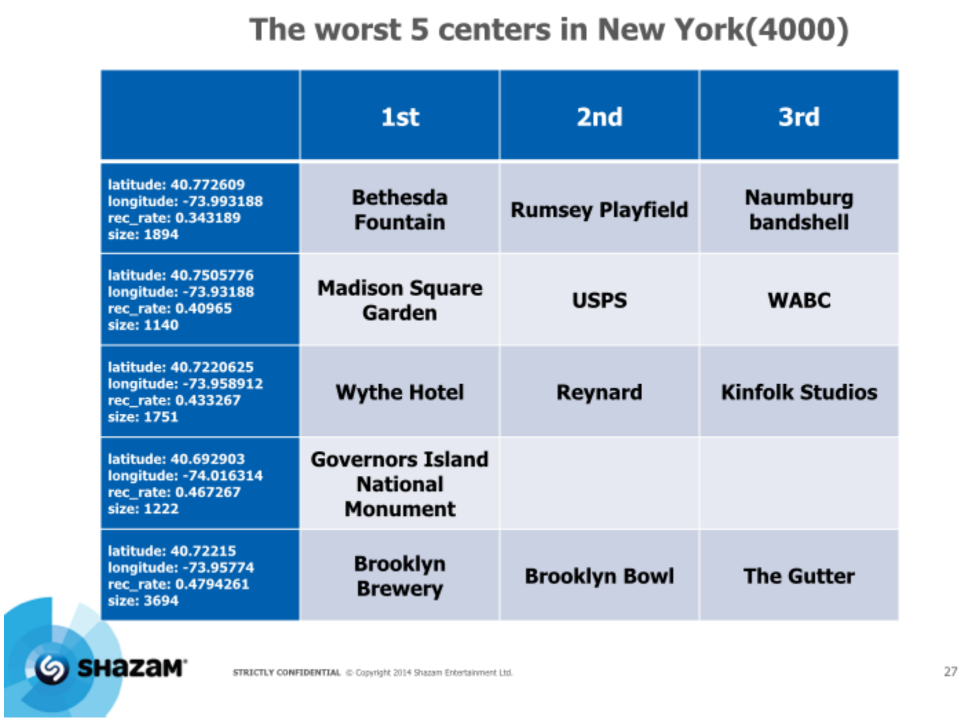
The worst 5 centers in London(3000)

	1st	2nd	3rd
latitude: 51.45966348 longitude: -0.1473565 rec_rate: 0.341151386 size: 938	Clapham Common Bandstand	Association of Capoeria Tooting	La Baita
latitude: 51.4955089 longitude: 0.00473804 rec_rate: 0.42204301 size: 1488	Brenntag UK Limited	O'Keefe	FIRST AID KITS AND TRAINING HSE
latitude: 51.5569371 longitude: -0.280226 rec_rate: 0.44740177 size: 789	The SSE Arena, Wembley	Wembley Stadium	Hilton London Wembley
latitude: 51.543778 longitude: -0.02273507 rec_rate: 0.44756447 size: 1745	Copper Box Arena	CRATE BREWRY& PIZZERIA	The Yard Theatre
latitude: 51.5223787 longitude: -0.080517 rec_rate:0.4770642 size: 763	Dennis Severs' House	Chariots	NSPCC

STRICTLY CONFIDENTIAL © Copyright 2014 Shazam Entertainment Ltd. 22

Example screenshots:

Clapham Common Bandstand area Shazam office area n London

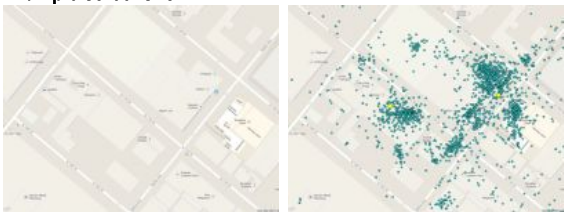


The worst 5 centers in New York(4000)

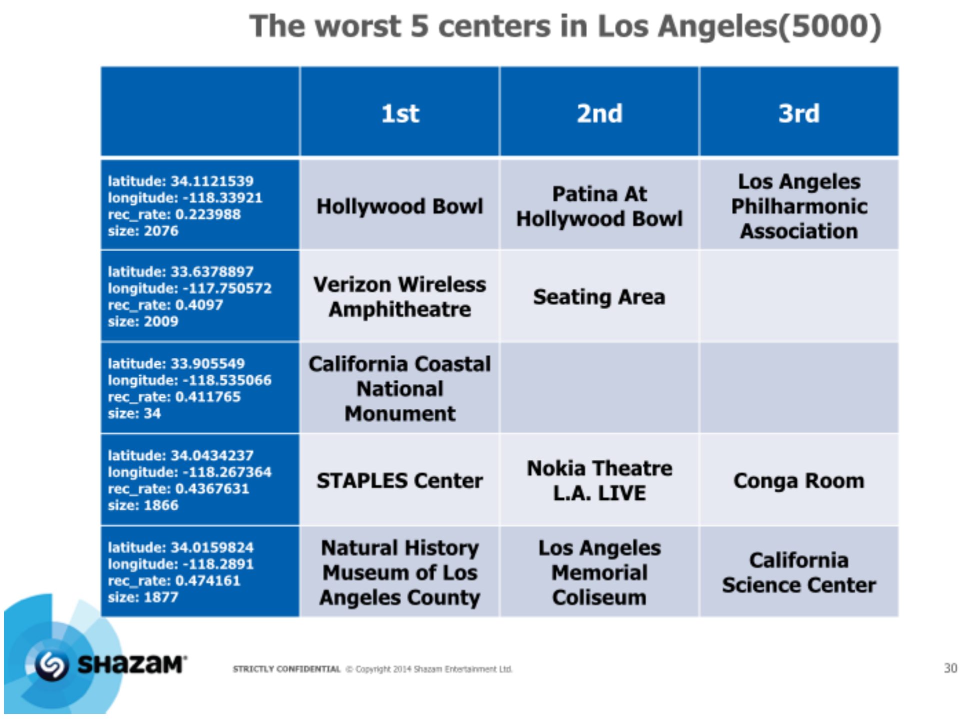
	1st	2nd	3rd
latitude: 40.772609 longitude: -73.993188 rec_rate: 0.343189 size: 1894	Bethesda Fountain	Rumsey Playfield	Naumburg bandshell
latitude: 40.7505776 longitude: -73.93188 rec_rate: 0.40965 size: 1140	Madison Square Garden	USPS	WABC
latitude: 40.7220625 longitude: -73.958912 rec_rate: 0.433267 size: 1751	Wythe Hotel	Reynard	Kinfolk Studios
latitude: 40.692903 longitude: -74.016314 rec_rate: 0.467267 size: 1222	Governors Island National Monument		
latitude: 40.72215 longitude: -73.95774 rec_rate: 0.4794261 size: 3694	Brooklyn Brewery	Brooklyn Bowl	The Gutter

27

Example screenshot:



Brooklyn Bars area (the third and fifth rows)



The worst 5 centers in Los Angeles(5000)

	1st	2nd	3rd
latitude: 34.1121539 longitude: -118.33921 rec_rate: 0.223988 size: 2076	Hollywood Bowl	Patina At Hollywood Bowl	Los Angeles Philharmonic Association
latitude: 33.6378897 longitude: -117.750572 rec_rate: 0.4097 size: 2009	Verizon Wireless Amphitheatre	Seating Area	
latitude: 33.905549 longitude: -118.535066 rec_rate: 0.411765 size: 34	California Coastal National Monument		
latitude: 34.0434237 longitude: -118.267364 rec_rate: 0.4367631 size: 1866	STAPLES Center	Nokia Theatre L.A. LIVE	Conga Room
latitude: 34.0159824 longitude: -118.2891 rec_rate: 0.474161 size: 1877	Natural History Museum of Los Angeles County	Los Angeles Memorial Coliseum	California Science Center

30

Example screenshot:

a cluster of a few points of interests (the fifth row)

Summary of the two methods and the best way I prefer to find potential POI

Both methods can figure out the recognition rate well for some certain situations, but since the geographical feature varies a lot and there are lots of different circumstances and extreme cases for the output of tags, they to some extent suffer one or more problems. I will list the cons and pros in the table below to let you have a good understanding of both methods:

	PROS	CONS
Shapefile method	1. straightforward 2. easy to summarize and identify (venues and types)	1. ideal shapefiles not available (there are lots of missing information about the name or type of the building and many venues have no corresponding polygon shapes to them) 2. can not find out the information about the outdoor area 3. kind of data-consuming(filter out lots of data outside the polygon) 4. geographical points shift problem(the latitude and longitude we got are not accurate enough) and cut out data close to polygons which both lead to bad accuracy
Kmeans method	1. cover all venues 2. identify cluster places instead of only certain building 3. fully use of data	1. hard to make analysis for the result(identify some certain good or bad place or summarize the types related to them) 2. the performance rely on the parameter a lot(the number of clusters in Kmeans: 'K', the radius and keyword we use from google API) 3. the drawbacks of Kmeans itself(not stable, sometimes randomness; can not filter out extreme points) 4. the accuracy problem which we can only accurately identify the bad cluster but not a certain venue(need common sense to get better result)

So the best way to figure out the recognition rate is to deal with the tradeoff of the two methods and fully use the advantages of both methods, I will prefer to do it this way:

1. Use the Kmeans clustering method to identify bad/good clusters(sorted by recognition rate), since the Kmeans method is time-consuming and also suffer from the noise problem we make do the clustering place by place instead of go through a rather larger part like state level, country level or even the worldwide level, the unsupervised learning method performs better at city level or more grained level
2. With the centers' latitude and longitude got from 1 we can pay our attention to the first/last several places like the worst 5 clusters in the area we choose
3. As for this step, we will find the potential bad/good places with the centers. I will use both the automatic way(google places API) and the manual method(look for places on google map). The google places API can help us find prominent places within a circle whose radius is usually 100 0r 200m, actually there is not a perfect value for all the cases, so at the same time we will also check the center with matched and unmatched tags on QGIS manually to find more accurate places, as for the final potential places we should also use our common sense to help us a little, for example the bars, stadiums or theaters are those kinds of places likely with low recognition rate.
4. To verify the rec-rate for a certain place, we can use a bounding box to cover the potential venue we are interested in and use the manullyverify_rec_rate.r to verify the correctness. The method is similar to shapefile method, so all in all it's a combination of two methods to get a more accurate result.
5. After all, if we can find more ideal shapefiles and can also deal with the accuracy of GPS problems good enough, we can directly use the shapefile method and combine the the results and the above results together to give a rather perfect report!!

Some details inside and other work I did

- When I do the kmeans clustering method, I kind of want to clean out those tags on the road which are usually with some speed in cars because we are more interested in the recognition rate in venues this time not on the road. So when I get the data from ec2, I filter out those data that have velocity feature and the speed is bigger than 2 miles per hour. However, only IOS users have the velocity feature and even lots of them are not available to us, so the improvement of the data-cleaning work is just so-so.
- As for the recognition rate in the cars, we can simply get the data with speed more than a certain threthold(I set it 1 mile/hour). After the calculation, I get the recognition rate with speed more than 1 is about 0.87 which fit for our common sense. I also check the relationship between the speed and recognition rate then, however there is no obvious relationship between these two variables. The related code is speed.r.
- There are various shapefiles online, but as I said before, none of them are ideal enough. The state level shapefile I found on Geofabrik contains only three features: id, name and type for each building which are all we need and the information given by it is easy to clean and process so I choose the California State level shapefile as an example to do the recognition rate calculation.
- The google places API has many parameters, two of the most important features in the nearby search requests are rankby and radius. I choose the radius as 200 meters for the output. The value 200 can make sure the output contain all the potential places around the center, but at the same time a larger area can also lead to the accuracy problem. As a result, we should tune the parameter for different situations, there is not a unique good value for it. As for the rankby keyword, there are two options: rankby = prominence and rankby = distance. I choose the rankby = prominence as default for two reasons: 1. it's convenient to use prominence since when you choose rankby = distance you have to specify

name, type or keyword. 2.we are more interested in prominent places which will likely be missed out by rankby = distance. However, for some certain situations , rankby = distance maybe have better and more accurate result than the prominence, so use common sense to deal with the tradeoff is better.
- In the script google_API.py, I filtered out the regional type like country, administrative in order to make the noise of the output less. I also clean out the type like park, since it's a larger area and the potential places are always some venues or some outdoor stadium inside the park not the park itself.
- In the kmeans clustering method, I use the center for the unmatched tags in each cluster instead of the center of the cluster in order to make it more accurate to find bad places. The way to do that can only improve our method, although I found the improvement is rather slight, whatever.
- I struggle for two weeks about modifying a pig embedded in python script with java UDF in it for KMEANS. I finally make it but the performance is bad : 6 mins per iteration. I am very sorry to give up the idea to run the script on the EMR, but I indeed learn a lot through the process. The code is here in case you are interested in:kmeans.py the other dependency of it is within my source code zipfile: sourcecode.zip
- During the first few weeks I also spend a lot of time on finding, viewing, editing, doing analysis about the shapefiles, the websites below are where I get the certain types of shapefiles and I found them useful, and the useful tool QGIS can also be downloaded here: QGIS

Shapefiles for certain points of interests: http://geocommons.com/

Shapefiles for states: http://www.geofabrik.de/

Shapefiles for cities: https://mapzen.com/metro-extracts/

Shapefiles for regional level: https://www.census.gov/geo/maps-data/data/tiger-line.html