Exploratory Data Analysis On Geolocational Data

11914960/RITESHWAR PRASAD SINGH

KM016,SCHOOL OF COMPUTER SCIENCE AND ENGG.

LOVELY PROFESSIONAL UNIVERSITY

Exploratory data Analysis is used by Data Scientist to analyse and summarize their main characteristics, often data visualisation method.

As a part of Crio #ibelieveindoing program, I tried some Data analysis on the Geolocational Data using Python.

Introduction

This project involves the use of K-Means Clustering to find the best accommodation for students in Bangalore (or any other city of your choice) by classifying accommodation for incoming students on the basis of their preferences on amenities, budget and proximity to the location.

Implementing the project will take you through the daily life of a data science engineer - from data preparation on real-life datasets, to visualising the data and running machine learning algorithms, to presenting the results.

Food delivery apps aside, managers of restaurant chains and hotels can also leverage this information. For example, if a manager of a restaurant already knows the demographic of his current customers, they'd ideally want to open at a location where this demographic is at its highest concentration, ensuring short commute times to the location and more customers served.If potential hotel locations are being evaluated, a site that caters to a wide variety of tastes would be ideal, since one would want every guest to have something to their liking.

This project is a good start for beginners and a refresher for professionals who have dabbled in python / ML before. The methodology can be applied to any location of one's choosing, so feel free to innovate! Objective

This project involves the use of K-Means Clustering to find the best accommodation for students in Bangalore (or any other city of your choice) by classifying accommodation for incoming students on the basis of their preferences on amenities, budget and proximity to the location.

Project Context

Implementing the project will take you through the daily life of a data science engineer - from data preparation on real-life datasets, to visualising the data and running machine learning algorithms, to presenting the results.

In the fast-moving, effort-intense environment that the average person inhabits, It's a frequent occurrence that one is too tired to fix oneself a home-cooked meal. And of course, even if one gets home-cooked meals every day, it is not unusual to want to go out for a good meal every once in a while for social/recreational purposes. Either way, it's a commonly understood idea that regardless of where one lives, the food one eats is an important aspect of the lifestyle one leads.

Now, imagine a scenario where a person has newly moved into a new location. They already have certain preferences, certain tastes. It would save both the student and the food providers a lot of hassle if the student lived close to their preferred outlets. Convenience means better sales, and saved time for the customer.

Food delivery apps aside, managers of restaurant chains and hotels can also leverage this information. For example, if a manager of a restaurant already knows the demographic of his current customers, they'd ideally want to open at a location where this demographic is at its highest concentration, ensuring short commute times to the location and more customers served.If potential hotel locations are being evaluated, a site that caters to a wide variety of tastes would be ideal, since one would want every guest to have something to their liking.

This project is a good start for beginners and a refresher for professionals who have dabbled in python / ML before. The methodology can be applied to any location of one's choosing, so feel free to innovate!

Stages

The project consists of the following stages:

High-Level Approach

Fetch Datasets from the relevant locations (Data Collection)

Clean the Datasets to prepare them for analysis. (Data Cleaning via Pandas)

Visualise the data using boxplots. (Using Matplotlib /Seaborn /Pandas)

Fetch Geolocational Data from the Foursquare API. (REST APIs)

Use K-Means Clustering to cluster the locations (Using ScikitLearn)

Present findings on a map. (Using Folium/Seaborn)

The desired end result of this project is something like this:

Final Output

Applications

K-Means clustering is used in a variety of examples or business cases in real life, like:

Academic performance (grouping students by their learning rate)

Diagnostic systems (grouping system faults under various reasons)

Search engines (grouping search results)

Wireless sensor networks (Mapping networks)

The FourSquare API data can be used for:

Building a restaurant review app like Swiggy Zomato etc.

Supporting a ride sharing service like Uber Pool

Summary

Clustering is the task of grouping the elements such that observations of same group are more similar to each other than those in other group.

Affinity Propagation is a graph-based algorithm that assigns each observation to its nearest exemplar. Basically, all the observations “vote” for which other observations they want to be associated with, which results in a partitioning of the whole dataset into a large number of uneven clusters.

Geolocational Analysis is the analysis that processes Satellite images, GPS coordinates and Street addresses and apply to geographic models.

so let's start, I need to import the following packages.

```
import numpy as np

import pandas as pd

## for plotting

import matplotlib.pyplot as plt

import seaborn as sns

## for geospatial

import folium

import geopy

## for machine learning

from sklearn import preprocessing, cluster

import scipy

## for deep learning

import minisom
```

Fetch the data we need and set up your environment before you move on to data analysis.

```
from pandas.io.json import json_normalize

import folium

from geopy.geocoders import Nominatim

import requests

CLIENT_ID = "CLient ID" # your Foursquare ID

CLIENT_SECRET = "Client Secret key" # your Foursquare Secret

VERSION = '20200316'

LIMIT = 10000
```

Set up your query in such a way that you can check for residential locations in a fixed radius around a point of your choosing.

```
url =
'https://api.foursquare.com/v2/venues/explore?&client_id={}&client_secret={}&v={}&ll={},{}&radius
={}&limit={}'.format(

   CLIENT_ID,
```

```
    CLIENT_SECRET,
    VERSION,
    17.448372, 78.526957,
    30000,
    LIMIT)

results = requests.get(url).json()
```

parse the response data into a usable dataframe.

```
venues = results['response']['groups'][0]['items']
nearby_venues = json_normalize(venues)
```

We also need a count of grocery stores, restaurants, gyms etc. around each residential location. Form another query to get all these locations (fixed in a short distance around each residential location) and hit the endpoint again.

```
resta=[]
oth=[]
for lat,long in zip(nearby_venues['venue.location.lat'],nearby_venues['venue.location.lng']):
    url = 'https://api.foursquare.com/v2/venues/explore?&client_id={}&client_secret={}&v={}&ll={},{}&radius={}&limit={}'.format(
      CLIENT_ID,
      CLIENT_SECRET,
      VERSION,
      lat,long,
      1000,
      100)
    res = requests.get(url).json()
    venue = res['response']['groups'][0]['items']
    nearby_venue = json_normalize(venue)
    df=nearby_venue['venue.categories']

    g=[]
    for i in range(0,df.size):
```

```
    g.append(df[i][0]['icon']['prefix'].find('food'))
  co=0
  for i in g:
    if i>1:
      co+=1
  resta.append(co)
  oth.append(len(g)-co)

nearby_venues['restaurant']=resta
nearby_venues['others']=oth
nearby_venues
```

Drop the irrelevant values, handle the NaN values(if any) and summarise the results into a dataframe.

In order to define the right k, I shall use the Elbow Method: plotting the variance as a function of the number of clusters and picking the k that flats the curve.

```
f=['venue.location.lat','venue.location.lng']
X = nearby_venues[f]
max_k = 10
## iterations
distortions = []
for i in range(1, max_k+1):
  if len(X) >= i:
    model = cluster.KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10, random_state=0)
    model.fit(X)
    distortions.append(model.inertia_)
## best k: the lowest derivative
k = [i*100 for i in np.diff(distortions,2)].index(min([i*100 for i
   in np.diff(distortions,2)]))
## plot
fig, ax = plt.subplots()
```

```
ax.plot(range(1, len(distortions)+1), distortions)

ax.axvline(k, ls='--', color="red", label="k = "+str(k))

ax.set(title='The Elbow Method', xlabel='Number of clusters',

    ylabel="Distortion")

ax.legend()

ax.grid(True)

plt.show()
```

I am going to create the map with folium, a really convenient package that allows us to plot interactive maps without needing to load a shapefile. Each store shall be identified by a point with size proportional to its current staff and color based on its cost. I'm also going to add a small piece of HTML code to the default map to display the legend.

```
x, y = "lat", "long"

color = "restaurant"

size = "others"

popup = "venue.location.formattedAddress"

data = n.copy()


## create color column

lst_colors=["red","green","orange"]

lst_elements = sorted(list(n[color].unique()))


## create size column (scaled)

scaler = preprocessing.MinMaxScaler(feature_range=(3,15))

data["size"] = scaler.fit_transform(

        data[size].values.reshape(-1,1)).reshape(-1)


## initialize the map with the starting location

map_ = folium.Map(location=location, tiles="cartodbpositron",

          zoom_start=11)

## add points

data.apply(lambda row: folium.CircleMarker(

      location=[row[x],row[y]],popup=row[popup],
```

```
        radius=row["size"]).add_to(map_), axis=1)
## add html legend



## plot the map
map_
```

We can try with k = 6 so that the K-Means algorithm will find 6 theoretical centroids. In addition, I will identify the real centroids too (the closest observation to the cluster center).

```
k = 6
model = cluster.KMeans(n_clusters=k, init='k-means++')
X = n[["lat","long"]]
## clustering
dtf_X = X.copy()
dtf_X["cluster"] = model.fit_predict(X)
## find real centroids
closest, distances = scipy.cluster.vq.vq(model.cluster_centers_,
                 dtf_X.drop("cluster", axis=1).values)
dtf_X["centroids"] = 0
for i in closest:
    dtf_X["centroids"].iloc[i] = 1
## add clustering info to the original dataset
n[["cluster","centroids"]] = dtf_X[["cluster","centroids"]]
n
```

I added two columns to the dataset: “cluster” indicating what cluster the observation belongs to, and “centroids” that is 1 if an observation is also the centroid (the closest to the center) and 0 otherwise. Let’s plot it out:

```
## plot
fig, ax = plt.subplots()
sns.scatterplot(x="lat", y="long", data=n,
                palette=sns.color_palette("bright",k),
                hue='cluster', size="centroids", size_order=[1,0],
                legend="brief", ax=ax).set_title('Clustering (k='+str(k)+')')
```

```
th_centroids = model.cluster_centers_

ax.scatter(th_centroids[:,0], th_centroids[:,1], s=50, c='black',

        marker="x")
```

Affinity Propagation is quite convenient when you can't specify the number of clusters, and it's suited for geospatial data as it works well with non-flat geometry.

```
model = cluster.AffinityPropagation()
```

Independently from the algorithm you used to cluster the data, now you have a dataset with two more columns ("cluster", "centroids"). We can use that to visualize the clusters on the map, and this time I'm going to display the centroids as well using a marker.

```
x, y = "lat", "long"

color = "cluster"

size = "restaurant"

popup = "venue.location.formattedAddress"

marker = "centroids"

data = n.copy()

## create color column

lst_elements = sorted(list(n[color].unique()))

lst_colors = ['#%06X' % np.random.randint(0, 0xFFFFFF) for i in

          range(len(lst_elements))]

data["color"] = data[color].apply(lambda x:

            lst_colors[lst_elements.index(x)])

## create size column (scaled)

scaler = preprocessing.MinMaxScaler(feature_range=(3,15))

data["size"] = scaler.fit_transform(

           data[size].values.reshape(-1,1)).reshape(-1)

## initialize the map with the starting location

map_ = folium.Map(location=location, tiles="cartodbpositron",

              zoom_start=11)

## add points

data.apply(lambda row: folium.CircleMarker(

        location=[row[x],row[y]],

        color=row["color"], fill=True,popup=row[popup],
```

```
        radius=row["size"]).add_to(map_), axis=1)

## add html legend

legend_html = """<div style="position:fixed; bottom:10px; left:10px; border:2px solid black; z-index:9999; font-size:14px;"> <b>"""+color+""":</b><br>"""

for i in lst_elements:

    legend_html = legend_html+""" <i class="fa fa-circle

    fa-1x" style="color:"""+lst_colors[lst_elements.index(i)]+""""">

    </i> """+str(i)+"""<br>"""

legend_html = legend_html+"""</div>"""

map_.get_root().html.add_child(folium.Element(legend_html))

## add centroids marker

lst_elements = sorted(list(n[marker].unique()))

data[data[marker]==1].apply(lambda row:

        folium.Marker(location=[row[x],row[y]],

        draggable=False,  popup=row[popup] ,

        icon=folium.Icon(color="black")).add_to(map_), axis=1)

## plot the map

map_
```

Conclusion

It was very insightful and great learning journey through entire project creation. I was able to built a view of the project. In the entire process I learnt about many new methods that used in Exploratory Data Analysis. Also through this project I landed up writing my first blog which I had never thought.

I am sharing my code repo and web-app link below:

Github:

LINK: https://github.com/Ritwshwar421

Welcome for any suggestion and review.