Density-based Place Clustering in Geo-Social Networks

Submitted By -
Keerti Chaudhary (181CO226)
Shumbul Arifa (181CO152)

Introduction

- Clustering is a common task of data mining
- it divides a set of objects into groups
  - objects in the same group (called a cluster) are similar to each other
  - objects in different clusters are dissimilar
- In GeoSNs (eg. Facebook Places) , users are allowed to capture their geographic locations and share them by an operation named check-in.
- A checkin is a triplet ⟨u, p, time⟩ modeling the fact that user u visited place with point location p = ⟨x, y⟩ at a certain time

Introduction

1. Geosocial Network clustering deals with the unsupervised grouping of places into clusters and finds important applications in urban planning and marketing.
2. The information regarding the relation of people to the clustered sites is not considered by current spatial clustering models.
3. The paradigm of density-based clustering needs to be improvised for its application on sites in which visitors are users of a geosocial network.
4. In our project, we have used Bright Kite and Gowalla datasets to implement Geosocial Network data clustering using various clustering approaches,
    - Partitioning approaches like K-means
    - Hierarchical approach like BIRCH
    - Density-based approaches like DBSCAN, HDBSCAN, and OPTICS.
5. Later we have also implemented Voronoi-based graphs.
6. We have also evaluated the different clustering algorithms using evaluation metrics like the Silhouette score, Calinski Harabasz score, and Davies Bouldin score.

Clustering

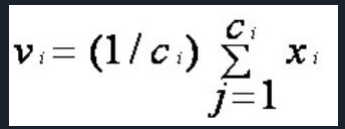
1. Clustering is a process that partitions a given data set into homogeneous groups based on given features such that similar objects are kept in a group whereas dissimilar objects are in different groups.
2. It is the most important unsupervised learning problem.
3. It deals with finding structure in a collection of unlabeled data.
4. For clustering algorithms to be advantageous and beneficial some of the conditions need to be satisfied.

- Scalability - Data must be scalable otherwise we may get the wrong result.
- The clustering algorithm must be able to deal with different types of attributes.
- A clustering algorithm must be able to find clustered data with an arbitrary shape.
- The clustering algorithm must be insensitive to noise and outliers.
- Interpret-ability and Usability - Results obtained must be interpretable and usable so that maximum knowledge about the input parameters can be obtained.

K-means algorithm

Let $X = \{x_1, x_2, x_3, \ldots\ldots, x_n\}$ be the set of data points and $V = \{v_1, v_2, \ldots\ldots, v_c\}$ be the set of centers.

1. Randomly select *'c'* cluster centers.
2. Calculate the distance between each data point and cluster centers.
3. Assign the data point to the cluster center whose distance from the cluster center is the minimum of all the cluster centers.
4. Recalculate the new cluster center using:



   a. where *'c_i'* represents the number of points in the *ith* cluster.
5. Recalculate the distance between each data point and newly obtained cluster centers.
6. If no data point was reassigned then stop, otherwise repeat from step 3).

Advantages - Fast , robust, easy to understand

Disadvantages - requires a priori specification of the number of cluster centers

BIRCH - algorithm

1) Building a clustering feature CFtree out of the data points, a height-balanced tree data structure.
2) The algorithm scans all the leaf entries in the initial CF tree to rebuild a smaller CF tree while removing outliers and grouping crowded subclusters into larger ones. This step is marked optional in the original presentation of BIRCH.
3) Here an agglomerative hierarchical clustering algorithm is applied directly to the subclusters represented by their CF vectors. It also allows the user to specify either the desired number of clusters or the desired diameter threshold for clusters. After this step, a set of clusters is obtained that captures major distribution patterns in the data.
4) The centroids of the clusters produced in step 3 are used as seeds and redistribute the data points to their closest seeds to obtain a new set of clusters.

Advantages - fast and do not need to know the expected number of clusters beforehand

Disadvantages - Favors only clusters with spherical shape and similar sizes and Handles only numeric data

DBSCAN - algorithm

Let $X = \{x_1, x_2, x_3, ..., x_n\}$ be the set of data points. DBSCAN requires two parameters: ε (eps) and the minimum number of points required to form a cluster (minPts).

1. Start with an arbitrary starting point that has not been visited.
2. Extract the neighborhood of this point using ε (All points which are within the ε distance are neighborhood).
3. If there are sufficient neighborhoods around this point then the clustering process starts and the point is marked as visited else this point is labeled as noise (Later this point can become part of the cluster).
4. If a point is found to be a part of the cluster then its ε neighborhood is also the part of the cluster and the above procedure from step 2 is repeated for all ε neighborhood points. This is repeated until all points in the cluster are determined.
5. A new unvisited point is retrieved and processed, leading to further cluster or noise discovery.
6. This process continues until all points are marked as visited.

Advantages - Does not require a priori specification of the number of clusters and detects noise

Disadvantages- fails in the case of varying density clusters

HDBSCAN - algorithm

1. Construct the *k*-nearest-neighbors (*k*-NN) graph, with an undirected edge connecting each point *p* to the *k* most similar points to *p*. Use the *k*-NN information to define a new metric called the *mutual reachability distance* between all pairs of points in the data.
2. Find the minimum spanning tree (MST) connecting all points in the data according to the mutual reachability distance. This tree represents a hierarchy of clusters, and the individual clusters can be extracted using a simple heuristic.

Advantages - built for the real-world scenario of having data with varying density and relatively fast

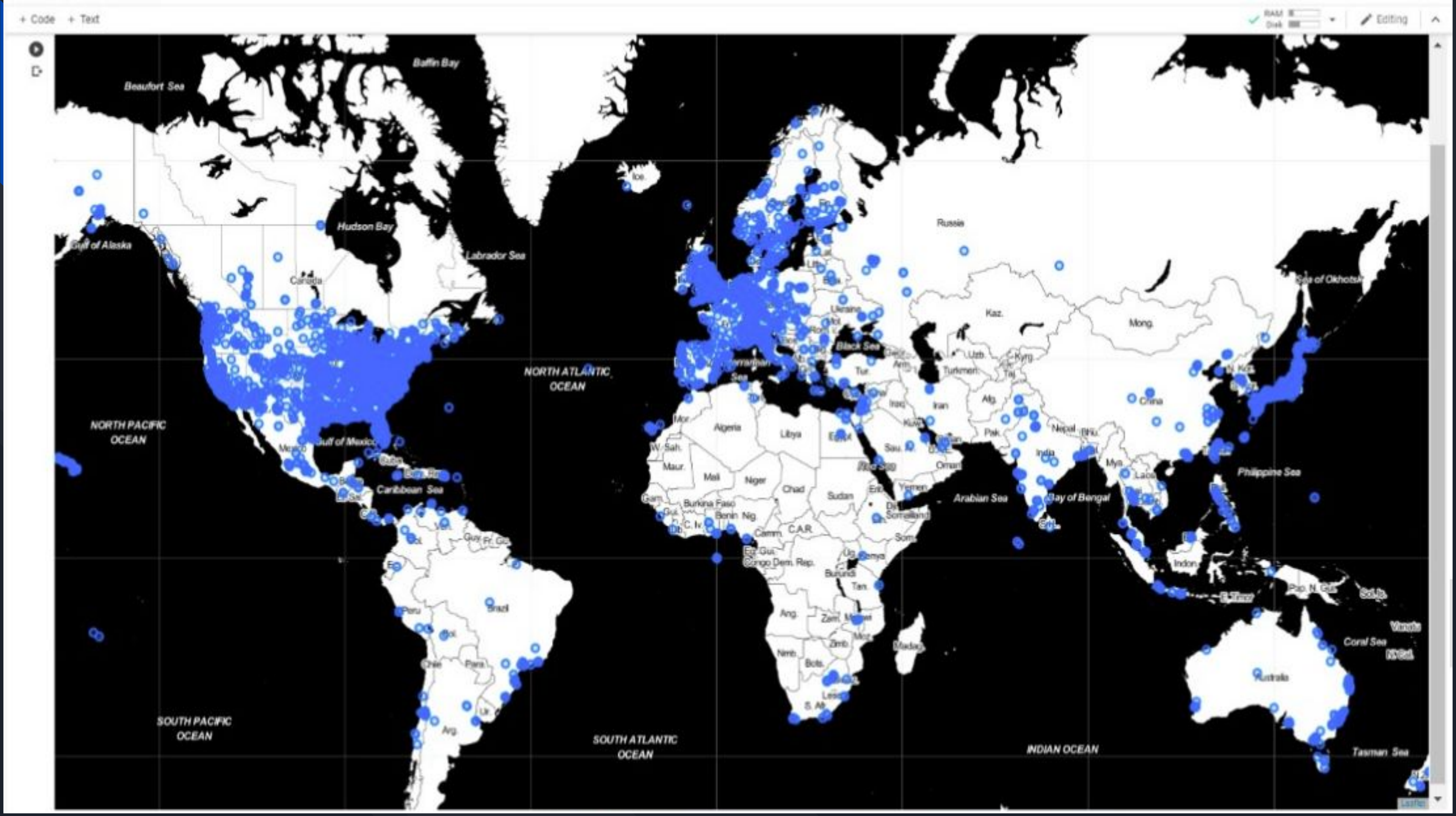
Disadvantages - not clear how to use it only with a subset of the data and Two dense data partitions on the bottom are not separated from the large clusters

OPTICS - algorithm

1. Mark every point in the dataset as unclustered. Corresponding to each point mark both core distance and reachability distance is undefined.
2. Pick any point and evaluate the neighborhood of that point.

Advantages - doesn’t require a predefined number of clusters and Clusters can be of any shape, including non-spherical ones

Disadvantages - It fails if there are no density drops between clusters


+ Code
+ Text
Editing
Baffin Bay
Beaufort Sea
Hudson Bay
Gulf of Alaska
Labrador Sea
Canada
NORTH ATLANTIC OCEAN
NORTH PACIFIC OCEAN
Gulf of Mexico
Caribbean Sea
Russia
Kaz.
Mong.
China
Black Sea
Iran
Algeria
Libya
Mali
Niger
Chad
Sudan
India
Nepal
Arabian Sea
Bay of Bengal
Philippine Sea
Sea of Okhotsk
Brazil
Peru
SOUTH PACIFIC OCEAN
SOUTH ATLANTIC OCEAN
INDIAN OCEAN
Tasman Sea
Coral Sea
Australia

Brightkite Dataset

```
5  df.head()
```

	user	check-in time	latitude	longitude	location id
0	0	2010-10-17T01:48:53Z	39.747652	-104.992510	88c46bf20db295831bd2d1718ad7e6f5
1	0	2010-10-16T06:02:04Z	39.891383	-105.070814	7a0f88982aa015062b95e3b4843f9ca2
2	0	2010-10-16T03:48:54Z	39.891077	-105.068532	dd7cd3d264c2d063832db506fba8bf79
3	0	2010-10-14T18:25:51Z	39.750469	-104.999073	9848afcc62e500a01cf6fbf24b797732f8963683
4	0	2010-10-14T00:21:47Z	39.752713	-104.996337	2ef143e12038c870038df53e0478cefc

```
1  df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4747287 entries, 0 to 4747286
Data columns (total 5 columns):
 #   Column         Dtype
---  ------         -----
 0   user           int64
 1   check-in time  object
 2   latitude       float64
 3   longitude      float64
 4   location id    object
dtypes: float64(2), int64(1), object(2)
```

DBSCAN sklearn std library (sklearn.cluster.DBSCAN)

class sklearn.cluster.DBSCAN(eps=0.5, *, min_samples=5, metric='euclidean', metric_params=None, algorithm='auto', leaf_size=30, p=None, n_jobs=None)

Parameters:

1. **eps** ***float, default=0.5***

   The maximum distance between two samples for one to be considered as in the neighborhood of the other.

2. **min_samples** ***int, default=5***

   The number of samples (or total weight) in a neighborhood for a point to be considered as a core point.

3. **metric_params** ***dict, default=None***

Additional keyword arguments for the metric function.

4. **algorithm*{'auto', 'ball_tree', 'kd_tree', 'brute'}, default='auto'***

The algorithm to be used by the NearestNeighbors module to compute pointwise distances and find nearest neighbors.

5. **leaf_size** ***int, default=30***

Leaf size passed to BallTree or cKDTree. T

6. **p*float, default=2***

The power of the Minkowski metric to be used to calculate distance between points.

7. **n_jobs** ***int, default=None***

The number of parallel jobs to run.

Attributes:

1. **core_sample_indices_*ndarray of shape (n_core_samples,)***

Indices of core samples.

2. **components_*ndarray of shape (n_core_samples, n_features)***

Copy of each core sample found by training.

3. **labels_*ndarray of shape (n_samples)***

Cluster labels for each point in the dataset given to fit(). Noisy samples are given the label -1.

OPTICS sklearn std library (sklearn.cluster.OPTICS)

class sklearn.cluster.**OPTICS**(**min_samples=5*,*max_eps=inf*, *metric='minkowski'*, *p=2*, *metric_params=None*, *cluster_method='xi'*, *eps=None*, *xi=0.05*, *predecessor_correction=True*, *min_cluster_size=None*, *algorithm='auto'*, *leaf_size=30*, *n_jobs=None*)

Parameters:

1. **Min_samples : *int > 1 or float between 0 and 1, default=5***

The number of samples in a neighborhood for a point to be considered as a core point

2. **Max_eps : *float, default=np.inf***

The maximum distance between two samples for one to be considered as in the neighborhood of the other.

3. **Metric : *str or callable, default='minkowski'***

Metric to use for distance computation. Any metric from scikit-learn or scipy.spatial.distance can be used.

4. **p:** ***int, default=2***

Parameter for the Minkowski metric from **pairwise_distances**.

5. **Cluster_method :** ***str, default='xi'***

The extraction method used to extract clusters using the calculated reachability and ordering.

6. **Eps :** ***float, default=None***

The maximum distance between two samples for one to be considered as in the neighborhood of the other.

7. **xi :** ***float between 0 and 1, default=0.05***

Determines the minimum steepness on the reachability plot that constitutes a cluster boundary.

8. **predecessor_correction*****bool, default=True***

Correct clusters according to the predecessors calculated by OPTICS

Attributes :

1. **labels_ : *ndarray of shape (n_samples,)***

   Cluster labels for each point in the dataset given to fit(). Noisy samples and points which are not included in a leaf cluster of `cluster_hierarchy_` are labeled as -1.

2. **reachability_ :*ndarray of shape (n_samples,)***

   Reachability distances per sample, indexed by object order.

3. **ordering_ :*ndarray of shape (n_samples,)***

   The cluster ordered list of sample indices.

4. **core_distances_*ndarray of shape (n_samples,)***

   Distance at which each sample becomes a core point, indexed by object order.

Computing DBSCAN

```
model = DBSCAN(eps=0.01, min_samples=5).fit(X_test)
class_predictions = model.labels_
```

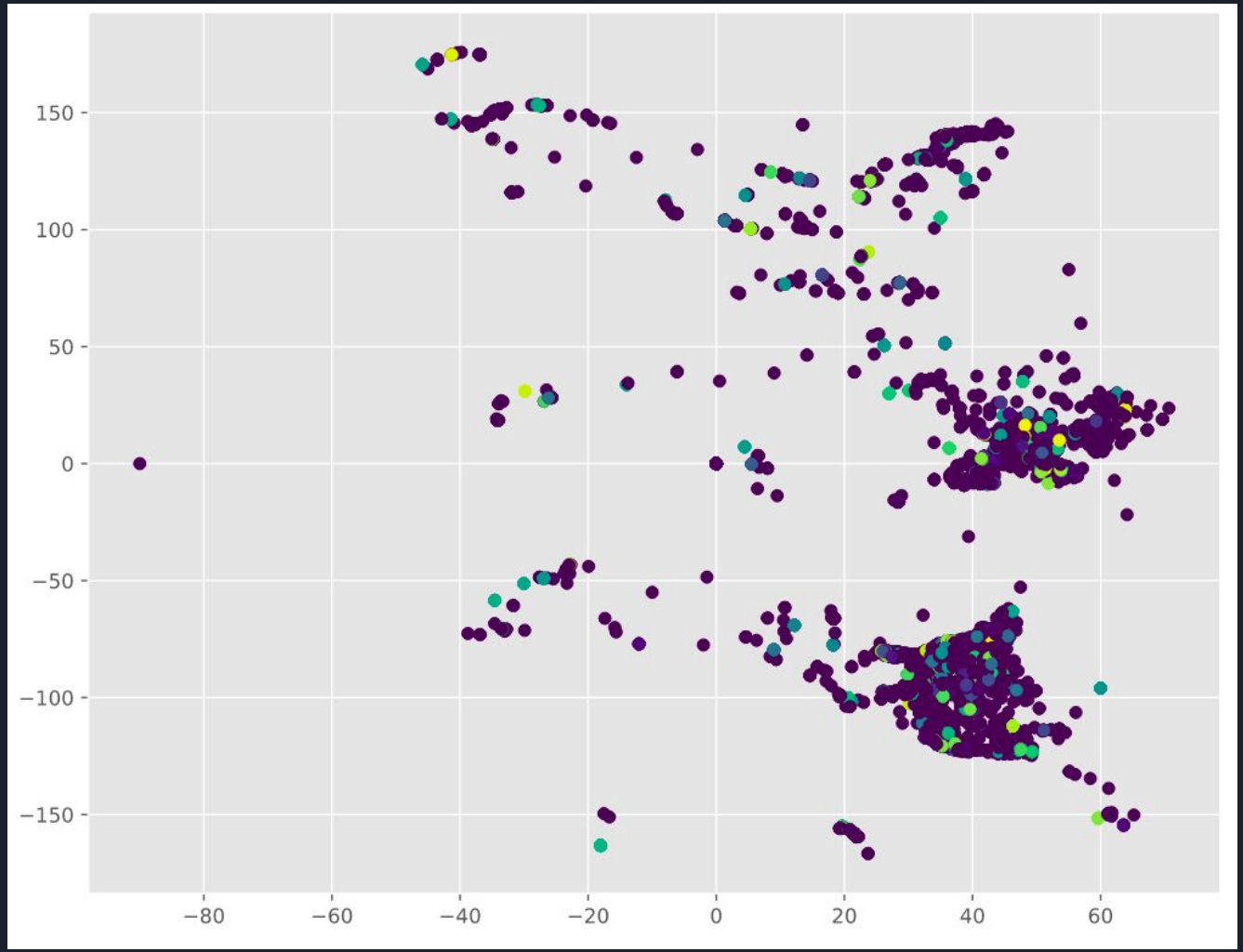
```
print(f'Number of clusters found: {len(np.unique(class_predictions))}')
print(f'Number of outliers found: {len(class_predictions[class_predictions==-1])}')

print(f'Silhouette ignoring outliers: {silhouette_score(X_test[class_predictions!=-1], class_predictions[class_predictic

no_outliers = 0
no_outliers = np.array([(counter+2)*x if x==-1 else x for counter, x in enumerate(class_predictions)])
print(f'Silhouette outliers as singletons: {silhouette_score(X_test, no_outliers)}')
```

```
Number of clusters found: 785
Number of outliers found: 11834
Silhouette ignoring outliers: 0.7661951430103257
Silhouette outliers as singletons: 0.2988007062481111
```

```
dbscan_dat=pd.DataFrame(X_test)
```


150
100
50
0
−50
−100
−150
−80
−60
−40
−20
0
20
40
60

Computing OPTICS

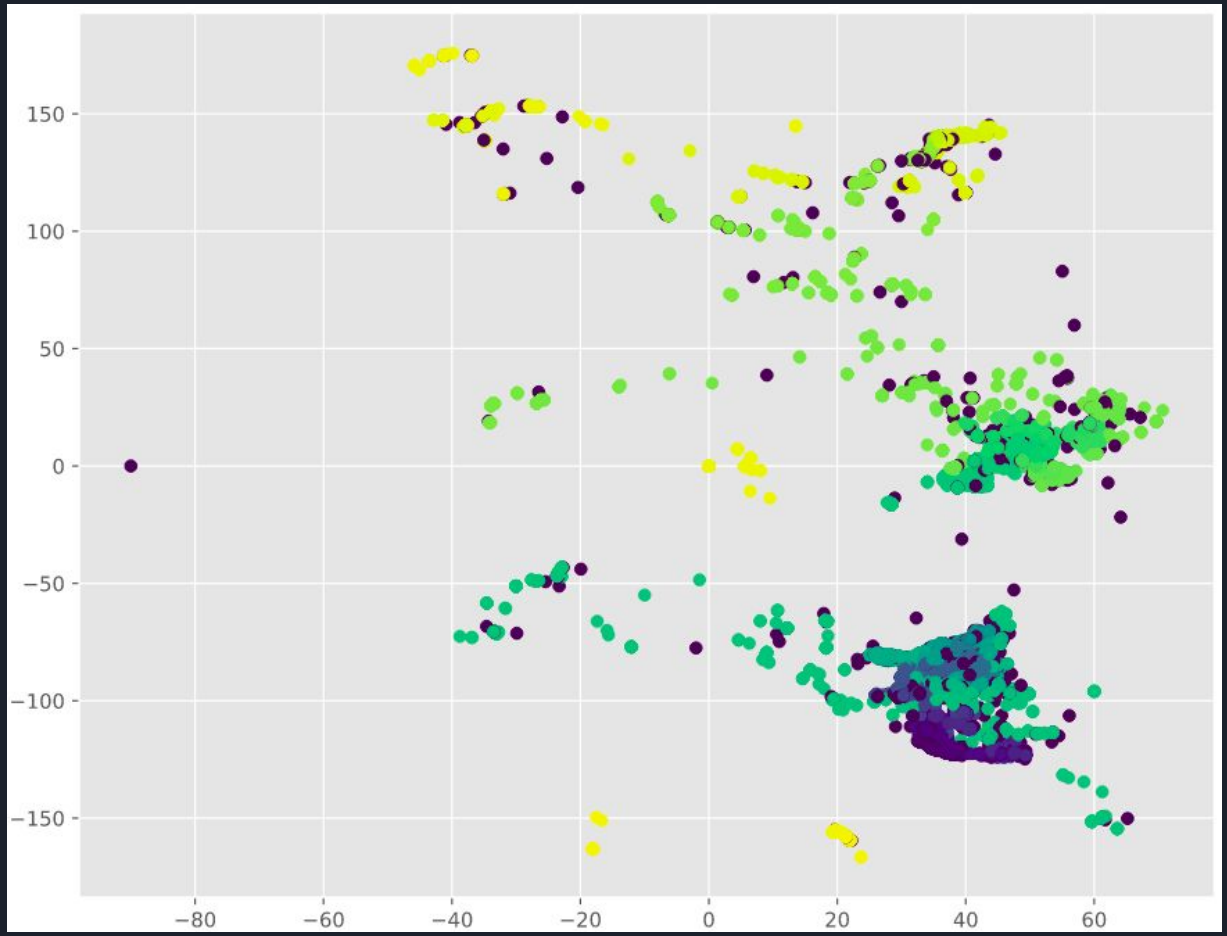
```
[ ]  1  model = OPTICS(min_samples=2)
```

```
[ ]  1  class_predictions= model.fit_predict(X_test)
```

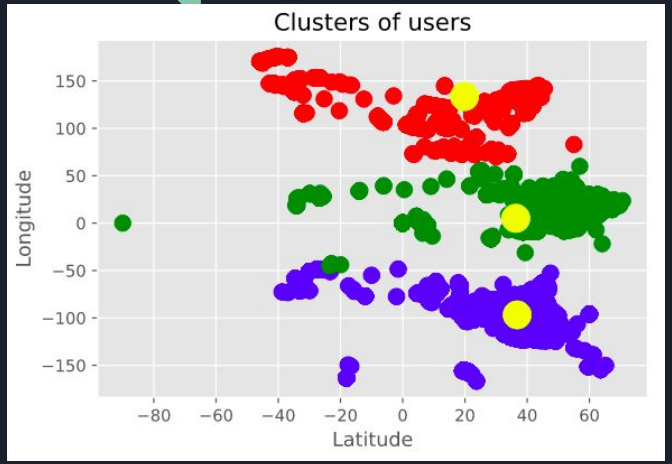
```
/usr/local/lib/python3.7/dist-packages/sklearn/cluster/_optics.py:802: RuntimeWarning: divide by zero encountered in true_divide
  ratio = reachability_plot[:-1] / reachability_plot[1:]
```

```
[ ]  1  print(f'Number of clusters found: {len(np.unique(class_predictions))-1}')
     2  print(f'Number of outliers found: {len(class_predictions[class_predictions==-1])}')
     3
     4  print(f'Silhouette ignoring outliers: {silhouette_score(X_test[class_predictions!=-1], class_predictions[class_predictions!=-1])}')
     5
     6  no_outliers = np.array([(counter+2)*x if x==-1 else x for counter, x in enumerate(class_predictions)])
     7  print(f'Silhouette outliers as singletons: {silhouette_score(X_test, no_outliers)}')
     8
     9  print(f'Calinski ignoring outliers: {calinski_harabasz_score(X_test[class_predictions!=-1], class_predictions[class_predictions!=-1])}')
    10
    11  no_outliers = np.array([(counter+2)*x if x==-1 else x for counter, x in enumerate(class_predictions)])
    12  print(f'Calinski outliers as singletons: {calinski_harabasz_score(X_test, no_outliers)}')
    13
    14  print(f'Davies Bouldin ignoring outliers: {davies_bouldin_score(X_test[class_predictions!=-1], class_predictions[class_predictions!=-1])}')
    15
    16  no_outliers = np.array([(counter+2)*x if x==-1 else x for counter, x in enumerate(class_predictions)])
    17  print(f'Davies Bouldin as singletons: {davies_bouldin_score(X_test, no_outliers)}')
```

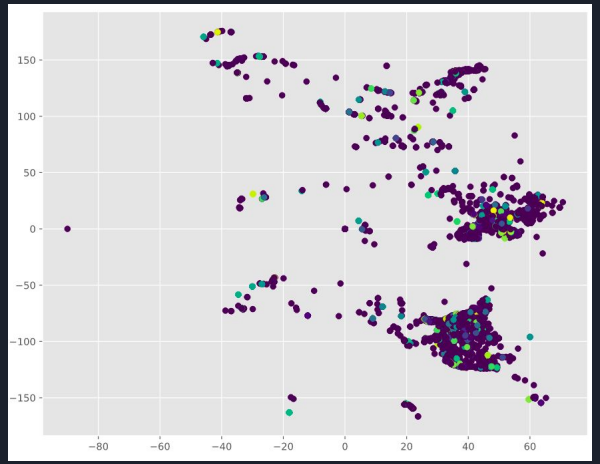
```
Number of clusters found: 9013
Number of outliers found: 5790
Silhouette ignoring outliers: 0.6622318262693725
Silhouette outliers as singletons: 0.5177249882562976
Calinski ignoring outliers: 915006.6944089212
Calinski outliers as singletons: 688000.7410681586
Davies Bouldin ignoring outliers: 0.3736579135589328
Davies Bouldin as singletons: 0.3229703827501595
```


150
100
50
0
−50
−100
−150
−80
−60
−40
−20
0
20
40
60

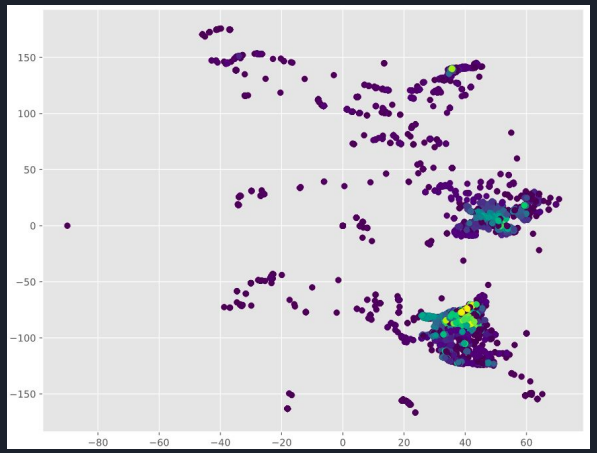
Scatter Plots



a. K-means

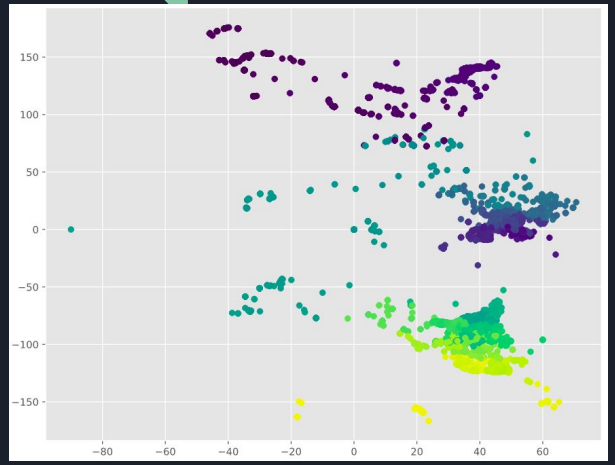


b. DBSCAN



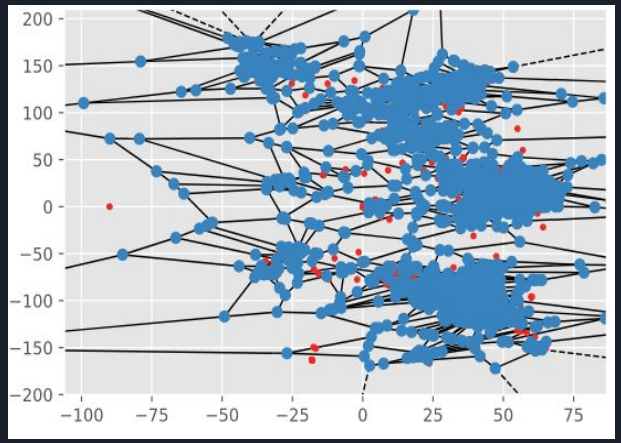
c. HDBSCAN

Scatter Plots



d. BIRCH

e. OPTICS



f. Voronoi

Evaluation metrics

- Silhouette score

$$\text{Silhouette score} = (p - q)/\max(p, q)$$

- Davies–Bouldin index

$$DB = \frac{1}{n}\sum_{i=1}^{n} max_{j \neq i}\left(\frac{\sigma_i + \sigma_j}{d(c_i, c_j)}\right)$$

- Calinski-Harabasz index

$$CH = \left[\frac{\sum_{k=1}^{K} n_k \|c_k - c\|^2}{K - 1}\right] / \frac{\sum_{k=1}^{K} \sum_{i=1}^{n_k} \|d_i - c_k\|^2}{N - K}$$

OBSERVATION

- Algorithm with the highest number of clusters = OPTICS
- Algorithm with lowest numbers of clusters = K-means
- Algorithm with highest Silhouette score = OPTICS
- Algorithm with lowest Silhouette score = HDBSCAN
- Algorithm with highest Calinski Harabasz score = DBSCAN
- Algorithm with lowest Calinski Harabasz score = K-means
- Algorithm with highest Davies Bouldin index = K-means
- Algorithm with lowest Davies Bouldin index = DBSCAN

Observations on the datasets

1) Based on the Silhouette score OPTICS Algorithm performs best
2) Based on Calinski Harabasz score DBSCAN Algorithm performs the best.
3) Based on Davies Bouldin score DBSCAN Algorithm performs the best.

THANK YOU