DATA MINING &
DATA WAREHOUSING

MINI PROJECT

Topic : Clustering

Submitted by:

Pavan Kumar Geddapu

BT16CSE067

CLUSTERING

Introduction

Clustering is the task of dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group and dissimilar to the data points in other groups. It is basically a collection of objects on the basis of similarity and dissimilarity between them.

There are many good clustering methods available in the python. The performance of those methods varies with the nature of the data we are dealing with. I tried to implement all the available clustering methods on a large public dataset which I explained in detail in the implementation part of this documentation.

Project Description

To apply various functions, algorithms for the clustering analysis available in the python libraries. Generate comparative statements on all of the methods used.

Software & Libraries Used

Jupyter notebook

Libraries:

- Pandas
- Sklearn
- pyclustering
- Numpy
- matplotlib

Project Implementation

Dataset

California housing dataset
This dataset was derived from the 1990 U.S. census, using one row per census block group. A block group is the smallest geographical unit for which the U.S. Census Bureau publishes sample data (a block group typically has a population of 600 to 3,000 people). This dataset is available in scikit-learn library.

Number of instances : 20640
Number of attributes : 8 numeric, predictive attributes and the target
Attribute information:

- MedInc : median income in block
- HouseAge : median house age in block
- AveRooms : average number of rooms
- AveBedrms : average number of bedrooms
- Population : block population
- AveOccup : average house occupancy
- Latitude : house block latitude
- Longitude : house block longitude

```
In [62]: # importing california housing data from sklearn datasets
         import pandas as pd
         from sklearn.datasets import fetch_california_housing

         housing_info = fetch_california_housing()
         data_mat = housing_info.data
         features = housing_info.feature_names
         data = pd.DataFrame(data_mat,columns=features)
         data
```

Out[62]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25
...	...	...	...	...	...	...	...	...
20635	1.5603	25.0	5.045455	1.133333	845.0	2.560606	39.48	-121.09
20636	2.5568	18.0	6.114035	1.315789	356.0	3.122807	39.49	-121.21
20637	1.7000	17.0	5.205543	1.120092	1007.0	2.325635	39.43	-121.22
20638	1.8672	18.0	5.329513	1.171920	741.0	2.123209	39.43	-121.32
20639	2.3886	16.0	5.254717	1.162264	1387.0	2.616981	39.37	-121.24

20640 rows × 8 columns

Fig 1 : California housing dataset

Preprocessing Data

The data imported has attributes of divergent range. To scale the data for better use I used one of the scikit-learn preprocessing algo called StandardScalar().

scaled_data = preprocessing.StandardScaler().fit_transform(data)

Next step is to reduce the dimensionality of the data for better visualization of the data and to increase the computational speed. I accomplished it using Principal Component Analysis (PCA) which is available in scikit-learn library. The dimensionality reduced to 2 from 8.

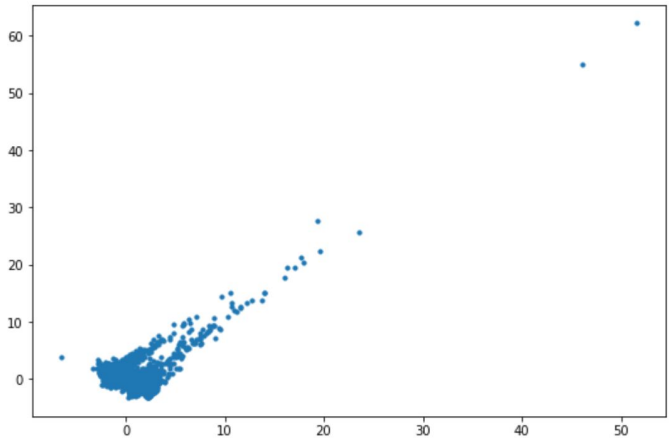

Fig 2 : data plot after reducing dimensions

As you can see this data contains some outliers (points on top corner etc..) which badly affect the later clustering process. So I removed those outliers and this is the output.

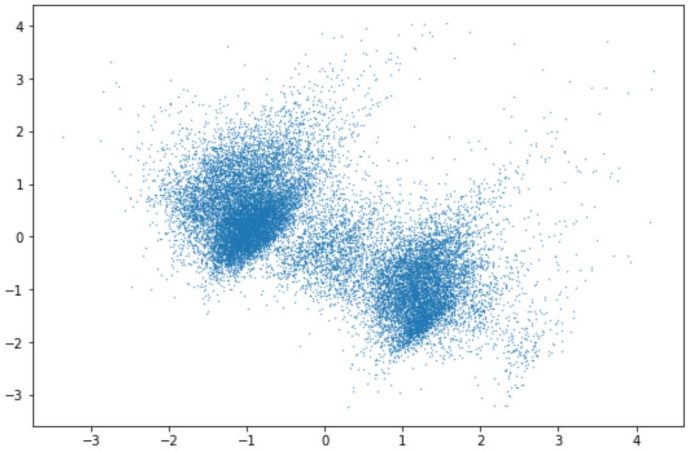

Fig 3 : data plot after removing outliers

Finally stored the processed data in a csv file.

Clustering Techniques

There are many good clustering techniques available in python . With the requirements the performance may vary among them. Here are some of those clusters which are readily available in python libraries.
The following are those methods with scatter plot depicting the cluster performance.

KMeans Clustering

KMeans algorithm is the widely used clustering technique. It is a partitioning technique. It is implemented via the KMeans class and the main configuration to tune is the “n_clusters” hyperparameter set to the estimated number of clusters in the data.
After several attempts of tuning the n_cluster parameter i found that value “2” giving the better output. I used scikit libraries KMeans method to accomplish this.

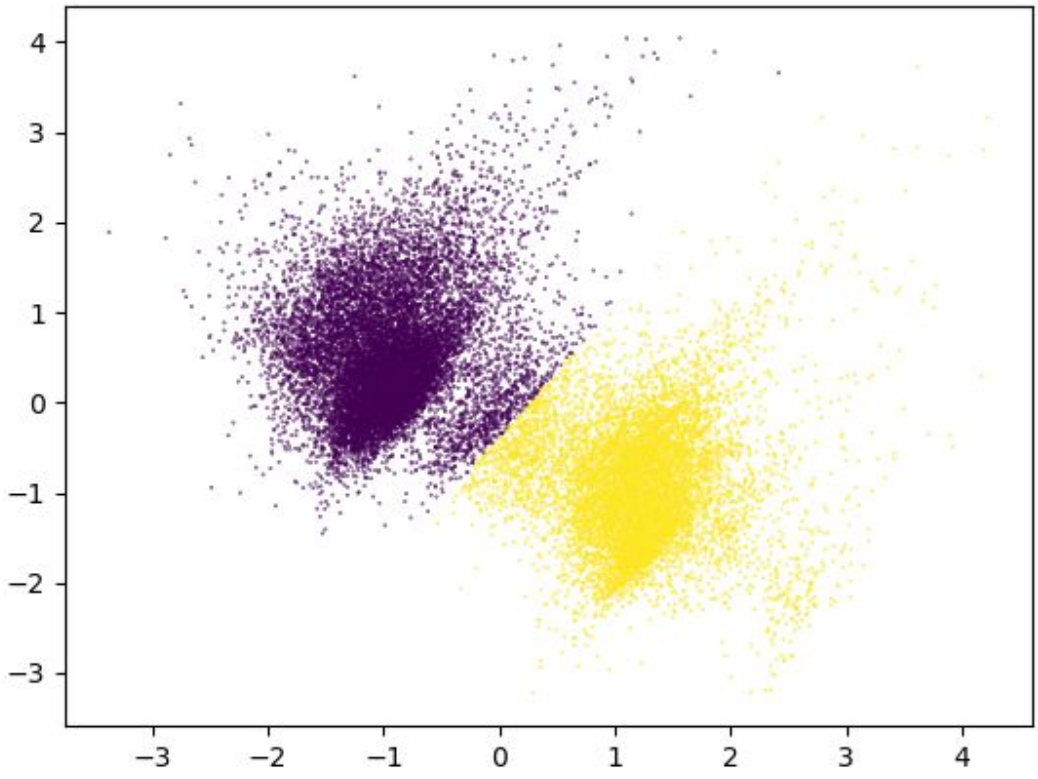

Fig 4 : KMeans Clustering

KMedoid Clustering

k -medoid is a classical partitioning technique of clustering, which clusters the data set of n objects into k clusters, with the number k of clusters assumed known a priori (which implies that the programmer must specify k before the execution of the algorithm).
As from the experience of KMeans I took the cluster count to be 2 for the better results.

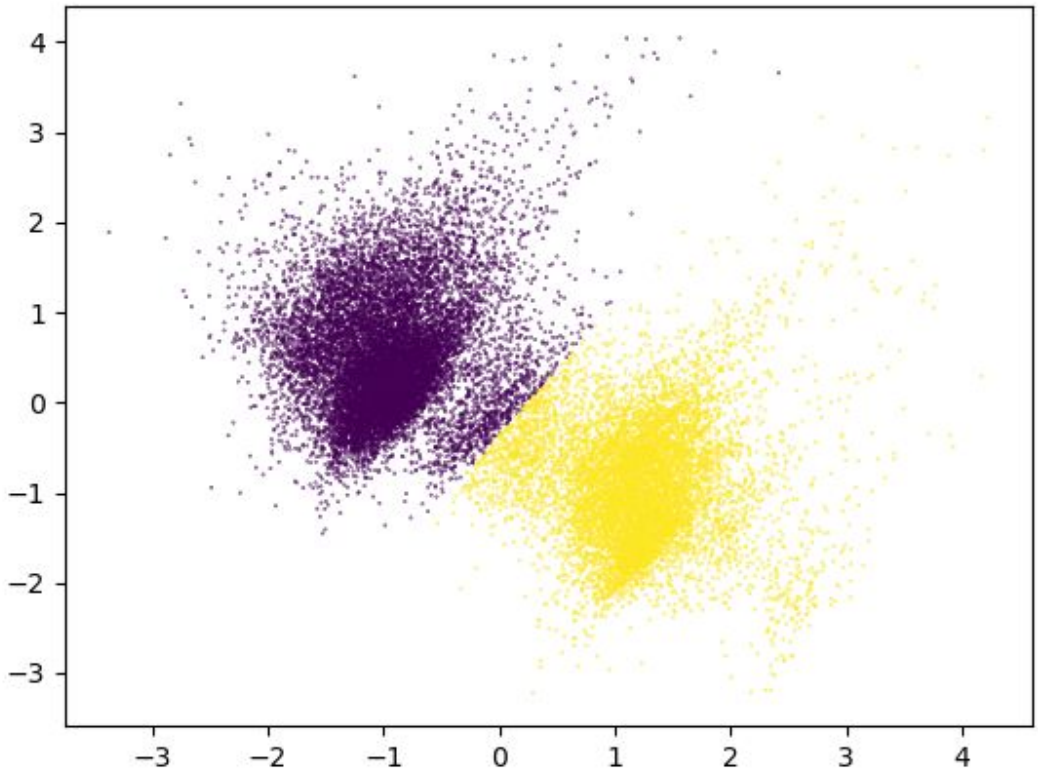

Fig 5: KMedoids clustering

I used KMedoids algorithm of “Pyclustering” library to accomplish this . It took more time to execute this method compared to KMeans. But the results look the same in both clustering techniques as they both belong to partitioning approach.

Hierarchical Clustering

I tried to partition the data in a hierarchical way using one of the methods called Ward and represented it as Dendrogram. It partitions the data in a tree format with respect to the distance between them.

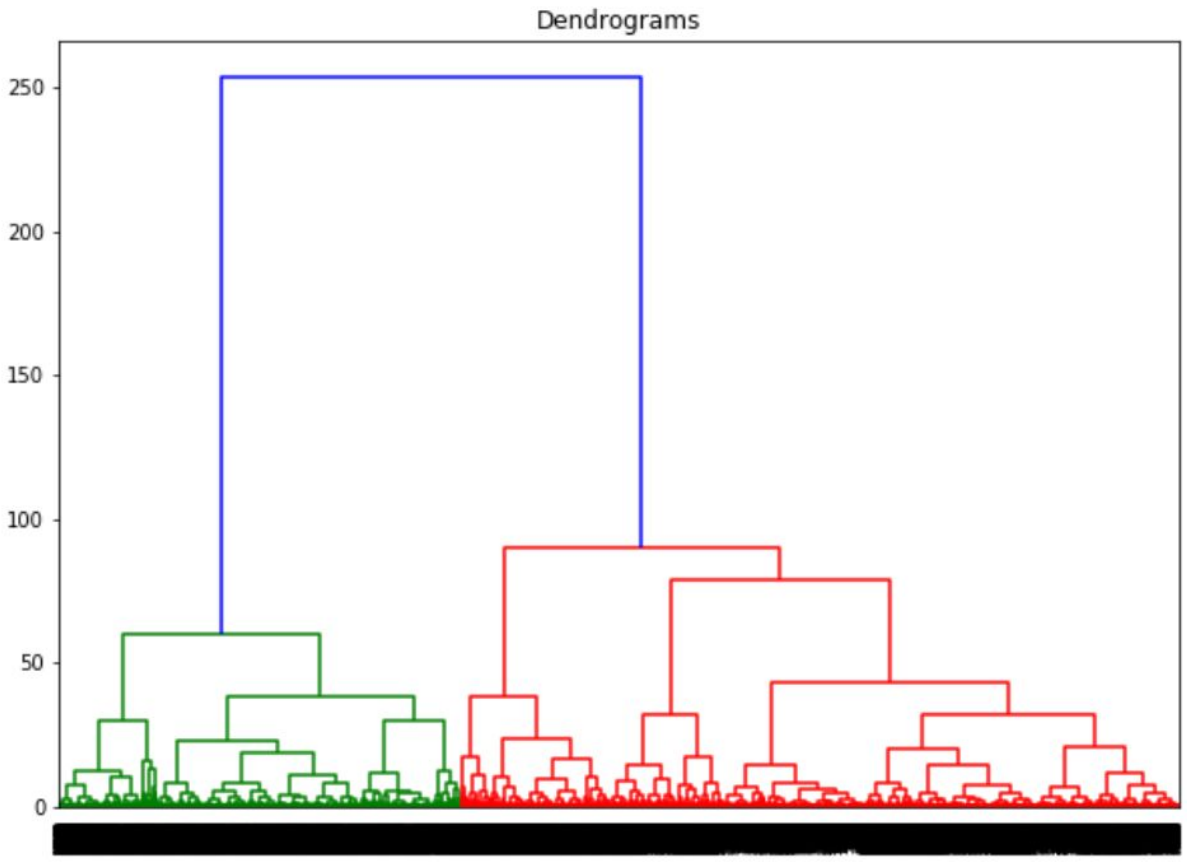

Fig 6: Dendrogram representing the data

The x-axis contains the data samples and y-axis represents the distance between them.
The vertical line with maximum length is the blue line so it is good to assume that the threshold i s 100 which partitions the data 2 clusters as we considered before. Even though this is the obvious cluster we can still partition the with reducing the threshold which may be helpful in future analysis.

Agglomerative Clustering (AGNES)

The agglomerative clustering is the most common type of hierarchical clustering used to group objects in clusters based on their similarity. It's also known as AGNES (Agglomerative Nesting). The algorithm starts by treating each object as a singleton cluster. Next, pairs of clusters are successively merged until all clusters have been merged into one big cluster containing all objects.

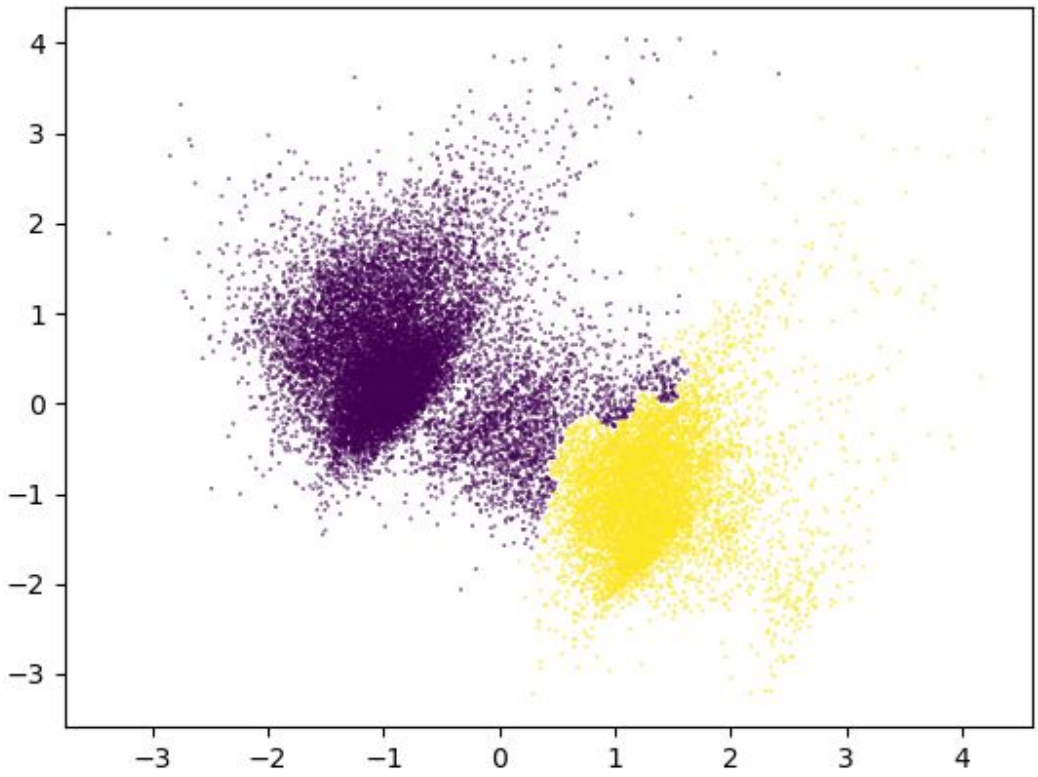

Fig 7: Agglomerative Clustering (AGNES)

This is a method in scikit-learn library . For this technique I need to provide n_cluster which represents how many clusters we are intending to partition the data. As from the previous page dendrogram it is obvious to take two clusters.

BIRCH

BIRCH (balanced iterative reducing and clustering using hierarchies) is an unsupervised data mining algorithm used to perform hierarchical clustering over particularly large data-sets. An advantage of BIRCH is its ability to incrementally and dynamically cluster incoming, multi-dimensional metric data points in an attempt to produce the best quality clustering for a given set of resources (memory and time constraints). In most cases, BIRCH only requires a single scan of the database.

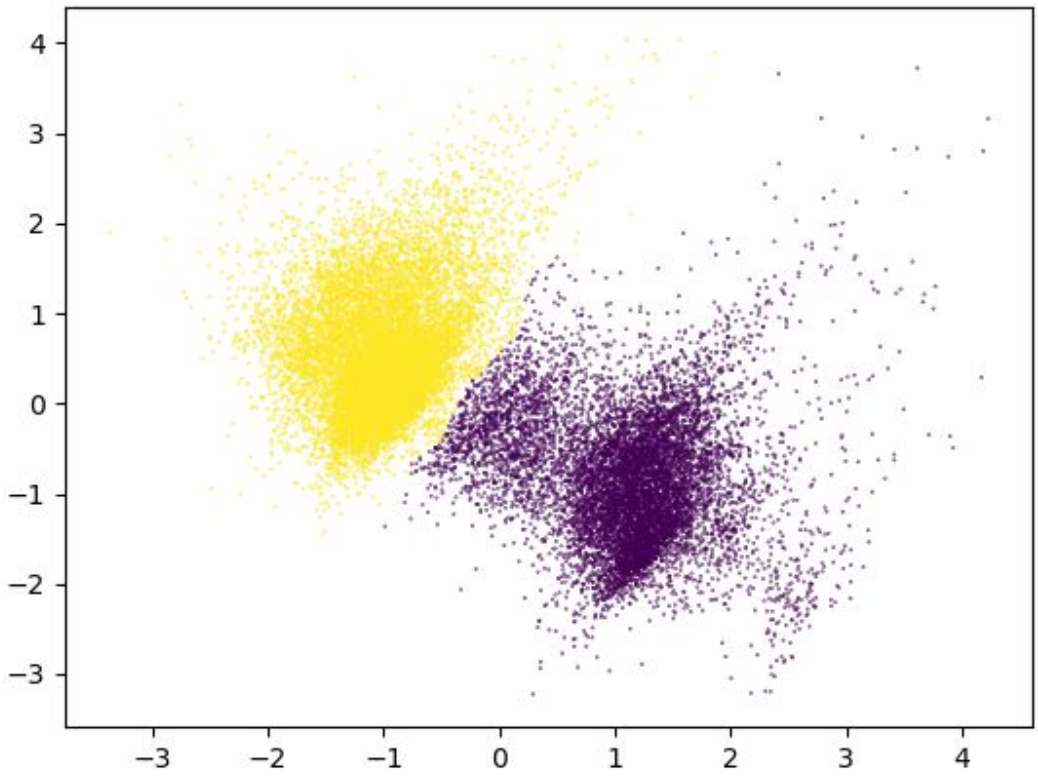

Fig 8 : BIRCH Method

This is a method in scikit-learn library . For this technique I need to provide n_cluster which represents how many clusters we are intending to partition the data. As from the previous page dendrogram it is obvious to take two clusters.

DBSCAN

Density-based spatial clustering of applications with noise (DBSCAN) is a data clustering algorithm proposed by Martin Ester, Hans-Peter Kriegel, Jörg Sander and Xiaowei Xu in 1996. It is a density-based clustering non-parametric algorithm: given a set of points in some space, it groups together points that are closely packed together (points with many nearby neighbors), marking as outliers points that lie alone in low-density regions (whose nearest neighbors are too far away).

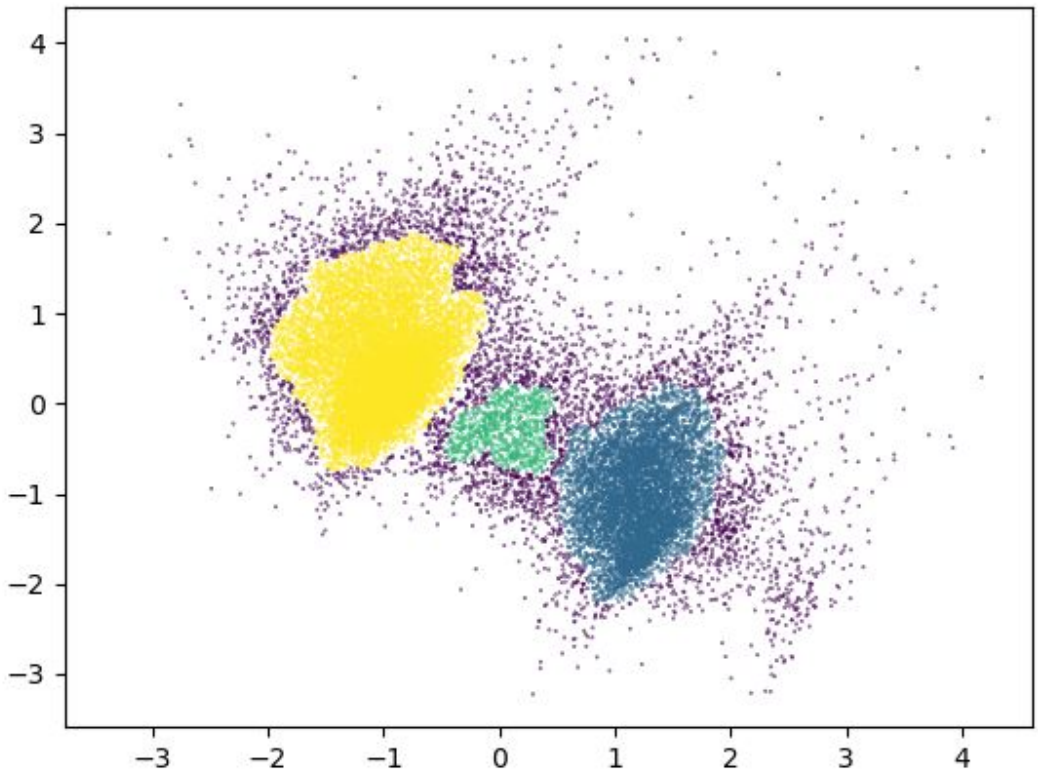

Fig 9 : DBSCAN

This is a method in scikit-learn library . In this method We need tune “eps (The maximum distance between two samples for one to be considered as in the neighborhood of the other), “min_sample (The number of samples in a neighborhood for a point to be considered as a core point)” parameters and i found eps=0.1 and min_samples=40 to be the best choice. This method gave 3 clusters and the violet data points may be considered as noise as they did not form any good cluster.

OPTICS

Ordering points to identify the clustering structure (OPTICS) is an algorithm for finding density-based clusters in spatial data. It was presented by Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel and Jörg Sander. Its basic idea is similar to DBSCAN, but it addresses one of DBSCAN's major weaknesses: the problem of detecting meaningful clusters in data of varying density. To do so, the points of the database are (linearly) ordered such that spatially closest points become neighbors in the ordering.

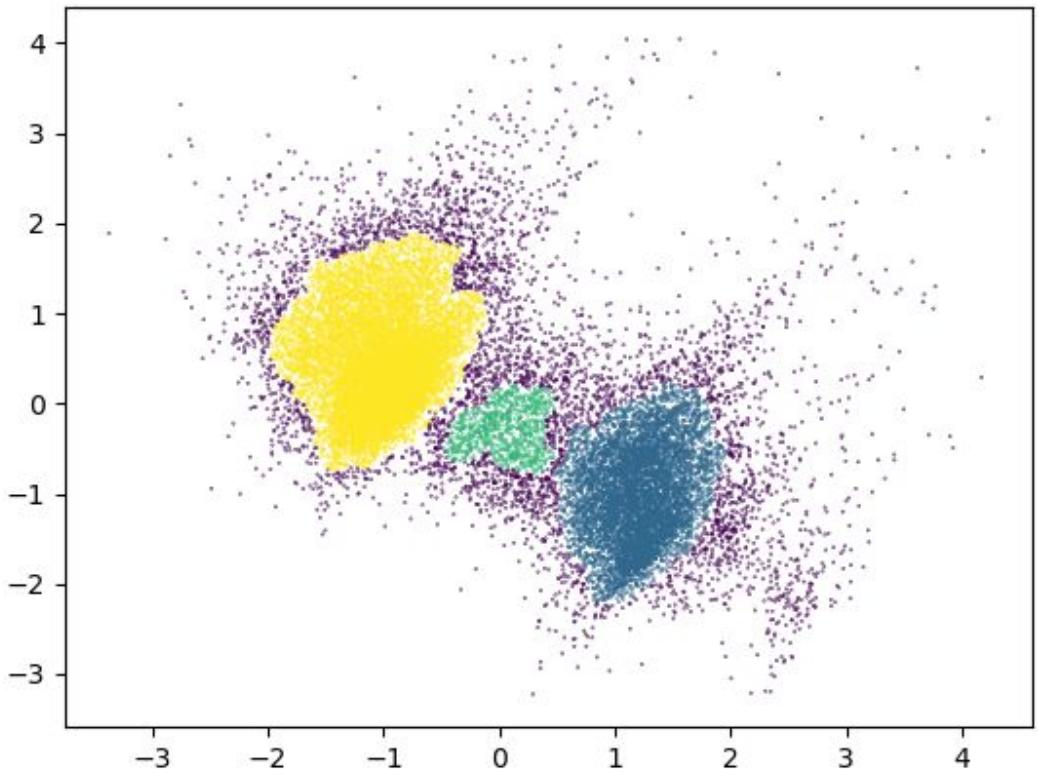

Fig 10 : OPTICS

This is a method in scikit-learn library . In this method We need tune “max_eps (The maximum distance between two samples for one to be considered as in the neighborhood of the other), “min_sample (The number of samples in a neighborhood for a point to be considered as a core point)” parameters and i found max_eps=0.1 and min_samples=40 to be the best choice. This method gave 3 clusters and the violet data points may be considered as noise as they did not form any good cluster.

CLIQUE

The CLIQUE Algorithm is a grid based approach. It finds clusters by first dividing each dimension into xi equal-width intervals and saving those intervals where the density is greater than tau as clusters. Then each set of two dimensions is examined: If there are two intersecting intervals in these two dimensions and the density in the intersection of these intervals is greater than tau, the intersection is again saved as a cluster. This is repeated for all sets of three, four, five,... dimensions. After every step adjacent clusters are replaced by a joint cluster and in the end all of the clusters are output.

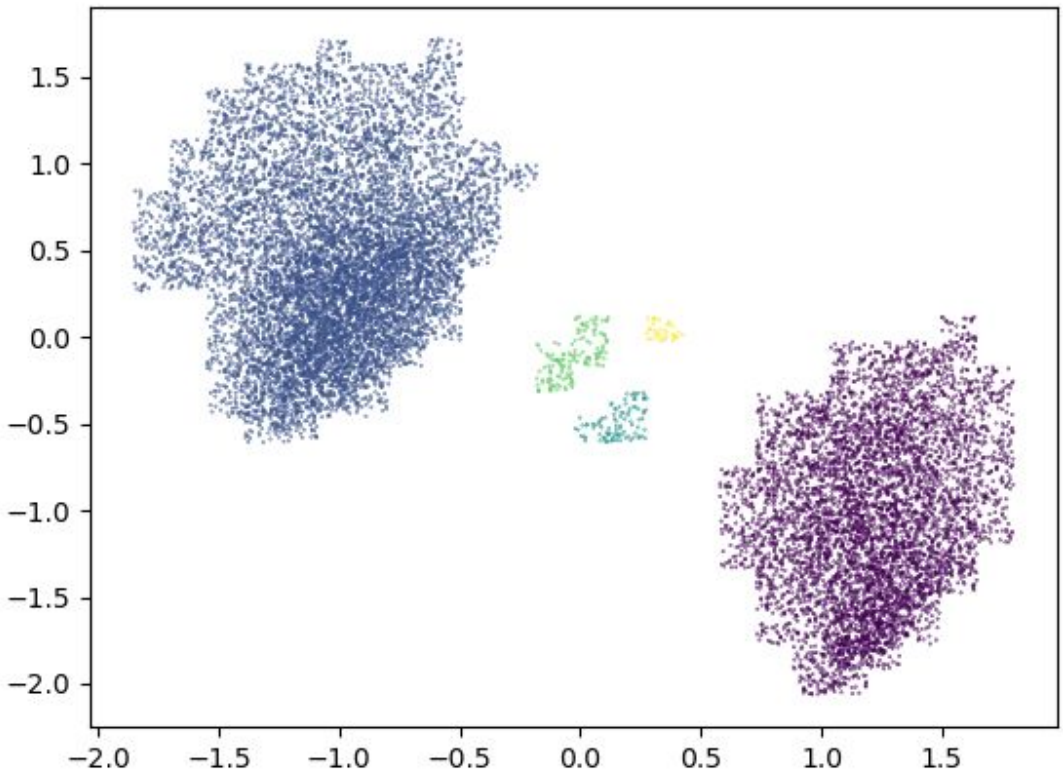

Fig 11 : OPTICS

This is a method in the pyclustering library. For good clustering we need to tune parameters “amount_intervals (Amount of intervals in each dimension that defines amount of CLIQUE block)”, “density_threshold (Minimum number of points that should contain CLIQUE block to consider its points as non-outliers)”. The remaining points not in the graph are removed, as they are considered as noise. It formed 5 clusters for amount_intervals=50 and density_threshold=30.

Spectral Clustering

Spectral clustering is a technique with roots in graph theory, where the approach is used to identify communities of nodes in a graph based on the edges connecting them. The method is flexible and allows us to cluster non graph data as well. It is implemented via the SpectralClustering class and the main Spectral Clustering is a general class of clustering methods, drawn from linear algebra. to tune is the “n_clusters” hyperparameter used to specify the estimated number of clusters in the data.

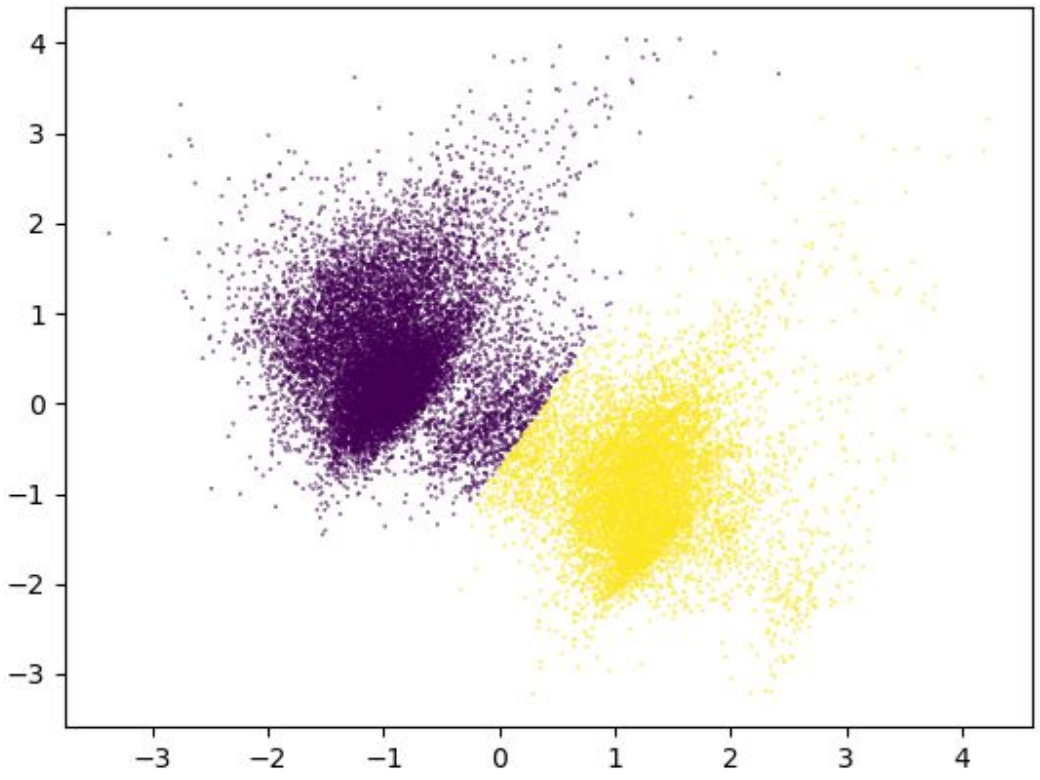

Fig 12 : Spectral Clustering

This is a method in scikit-learn library . For this technique I need to provide n_cluster which represents how many clusters we are intending to partition the data. I initiated n_cluster as 2 as it gave promising results compared to other values.

Mean Shift

Mean shift clustering aims to discover “blobs” in a smooth density of samples. It is a centroid-based algorithm, which works by updating candidates for centroids to be the mean of the points within a given region. These candidates are then filtered in a post-processing stage to eliminate near-duplicates to form the final set of centroids. Seeding is performed using a binning technique for scalability.

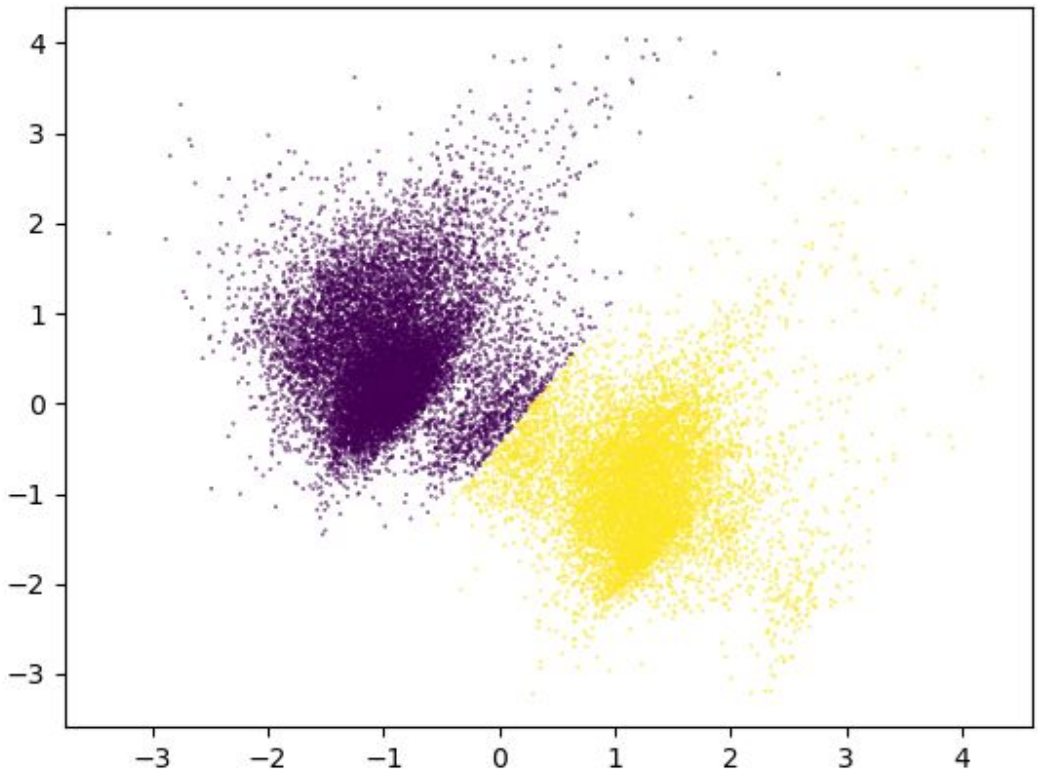

Fig 13 : Mean Shift Clustering

This is a method in scikit-learn library . We do not need to provide any parameters and it computes clusters using the data provided. As we can see from the above graph it partitioned the data into two clusters similar to KMeans and KMedoids.

Gaussian Mixture Model

A Gaussian mixture model summarizes a multivariate probability density function with a mixture of Gaussian probability distributions as its name suggests.

It is implemented via the GaussianMixture class and the main configuration to tune is the "n_clusters" hyperparameter used to specify the estimated number of clusters in the data.

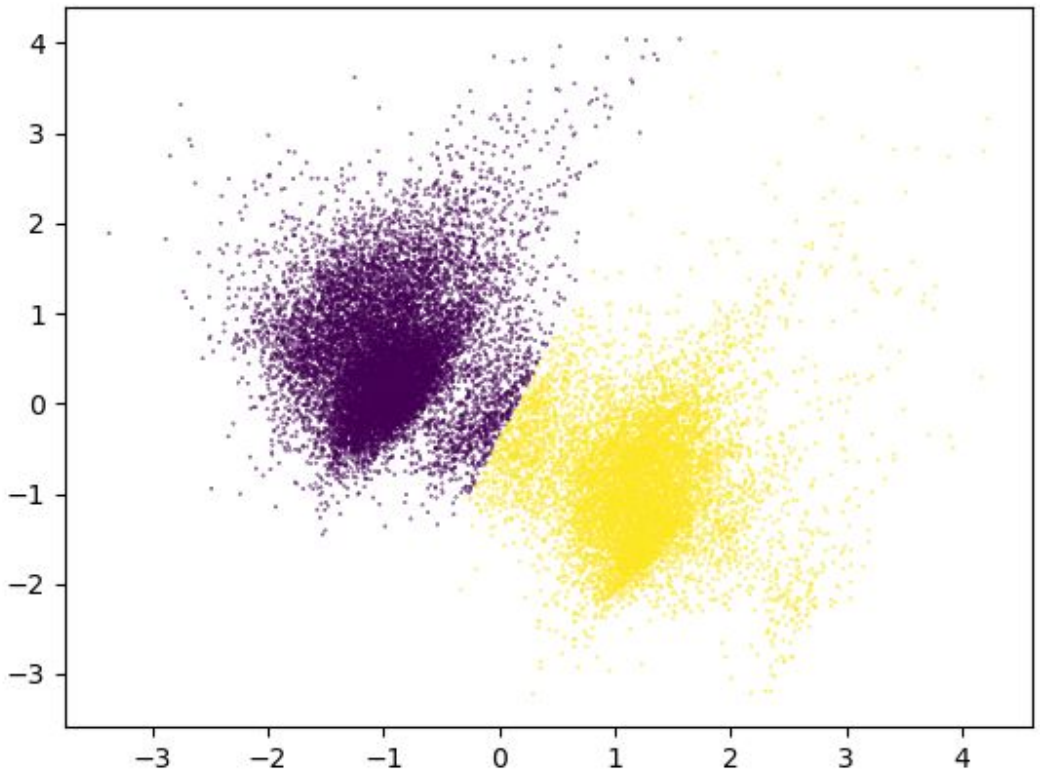

Fig 14 : Gaussian Mixture Model

This is a method in scikit-learn library . we need to provide the number of clusters as a parameter to the model. After several tries I found that n_clusters=2 is giving the best results. Even though it looks the same as KMeans we can observe slight differences in the boundaries of two clusters.

Comparing Different Clustering Algorithms

Algorithm Name	No. of Clusters	Time Taken (in seconds)
KMeans	2	0.11072086334228516
KMedoids	2	6.177213191986084
AGNES	2	21.157201528549194
BIRCH	2	0.5760328769683838
DBSCAN	3	0.508601016998291
OPTICS	3	20.75701379776001
CLIQUE	5	0.6779460906982422
Spectral Clustering	2	534.6829919815063
Mean Shift	2	286.0968792438507
Gaussian Mixture Model	2	0.086959896087646484

In Time Taken column

- Rows coloured in red indicate that those algorithms took a long time to converge.
- Rows coloured in blue are converged very fast
- Orange to indicates a moderate time

In No. of clusters column

- Most of the clusters are giving optimal results at no. of clusters = 2.
- Only one cluster giving optimal results at no. of clusters = 5.
- Two clusters are optimal at no. of clusters = 3.

Cluster Comparison Summary

After comparing all the above clusters these are my conclusions :

- Clusters KMeans, KMedoids give almost the same results with KMedoids taking a little bit more time to converge compared to KMean. With a clear look at the scatter plot it is obvious to partition them into 3 clusters but due to some outliers in the data, these algorithms are not able to cluster them to full extent.

- Hierarchical clusters AGNES, BIRCH gave results which nearly look the same with the exception that each one tries to align in different directions due to the implementation of those clusters. They both partitioned the data into 2 clusters. The AGNES algorithm took more time compared to BIRCH.

- Density based cluster algorithms DBSCAN, OPTICS gave results near to my expectation. As the data belongs to houses located in a span of area and most of the data is slightly inclined towards the area the house was situated in it is obvious that these algorithms did good work in partitioning them perfectly. Both of the algorithms partitioned the data into 3 clusters which i was expecting. Both algos scatter plots look nearly the same which proves my point. Though OPTICS algo took more time to converge the data compared to DBSCAN.

- The CLIQUE model is a Grid based approach. So it also performed well on clustering the data even though it removed a lot of data as noise. It further divided the middle cluster of density based algos into further clusters. This algorithm's execution is very fast compared to most of the other algorithms.

- Gaussian Mixture Model, Mean Shift, Spectral Clustering algorithms performed similar to algorithms of partitioning approach. The Gaussian Mixture Model is the fastest to execute in all the clustering algorithms. Even though Mean Shift, Spectral Clustering took a lot of time to execute all five algorithms (KMeans, KMedoids, Gaussian Mixture Model, Mean Shift, Spectral Clustering) gave similar results given the nature of the data.

Conclusion

- Considering the nature of the data the algorithms belonging to density based approach gave the best results (namely DBSCAN, OPTICS). Even the CLIQUE algorithm which belongs to a grid based approach gave results similar to DBSCAN and OPTICS but it removed a lot of data points as noise which is not optimal.

- Of all the algorithms Gaussian Mixture Model executed the fastest followed by KMeans. Spectral Algorithm took a lot of time and memory (nearly 3.13 GB of RAM) to execute though it gave results similar to KMeans.