Market Segmentation using Density-Based Clustering

Data Mining Project

Semester 4,
Academic Year 2021-22

01 – Density-Based Clustering

- Unsupervised learning methods
- Identify distinctive groups in the data – based on the idea that a cluster in a data space with similar values is a contiguous region of high point density.
- The data points in the separating regions of low point density are typically considered as noise (i.e. outliers)

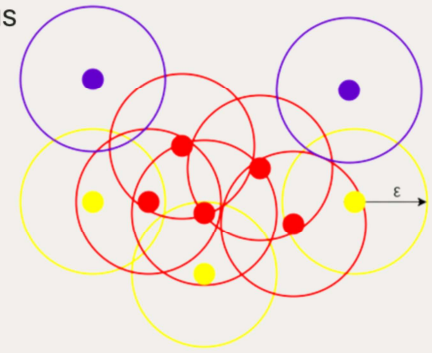
Yellow point here - Outlier

Unsupervised learning methods are when there is no clear objective or outcome we are seeking to find. Instead, we are clustering the data together based on the similarity of observations.

02 — DBSCAN

Density-Based Spatial Clustering of Applications with Noise

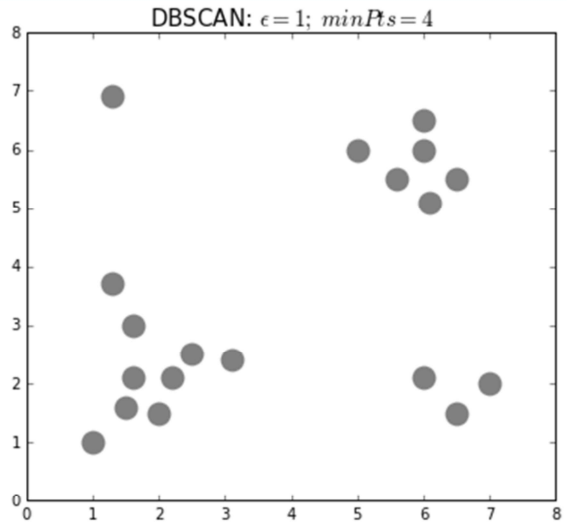
- The principle of DBSCAN is to find the neighborhoods of data points that exceed certain density threshold.
- The density threshold is defined by two parameters: the radius of the neighborhood (eps) and the minimum number of neighbors/data points (minPts) within the radius of the neighborhood.
- With these two thresholds in mind, DBSCAN starts from a random point to find its first density neighborhood.



contd.

02 – DBSCAN

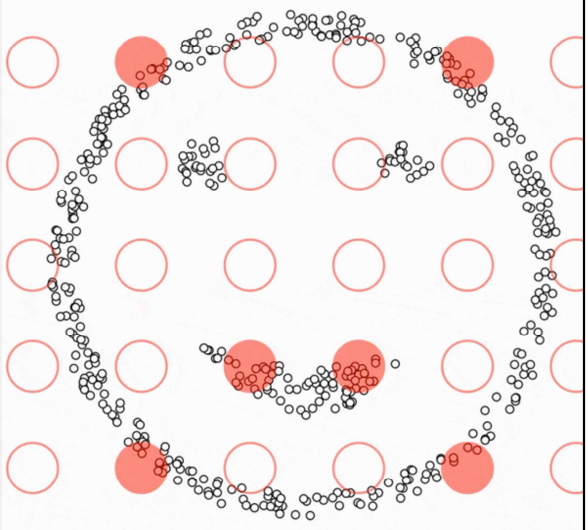
- Then from its first density neighborhood, it starts to find the second density neighborhood. If the second density neighborhood exists, DBSCAN will merge the first and second density neighborhoods to become a bigger density neighborhood.
- The same process keep going on until its neighborhood cannot meet the threshold anymore.
- The 'find and merge' process allows DBSCAN to find all the density neighborhoods from a random starting point and finally merge all of them into one final big neighborhood – a cluster.

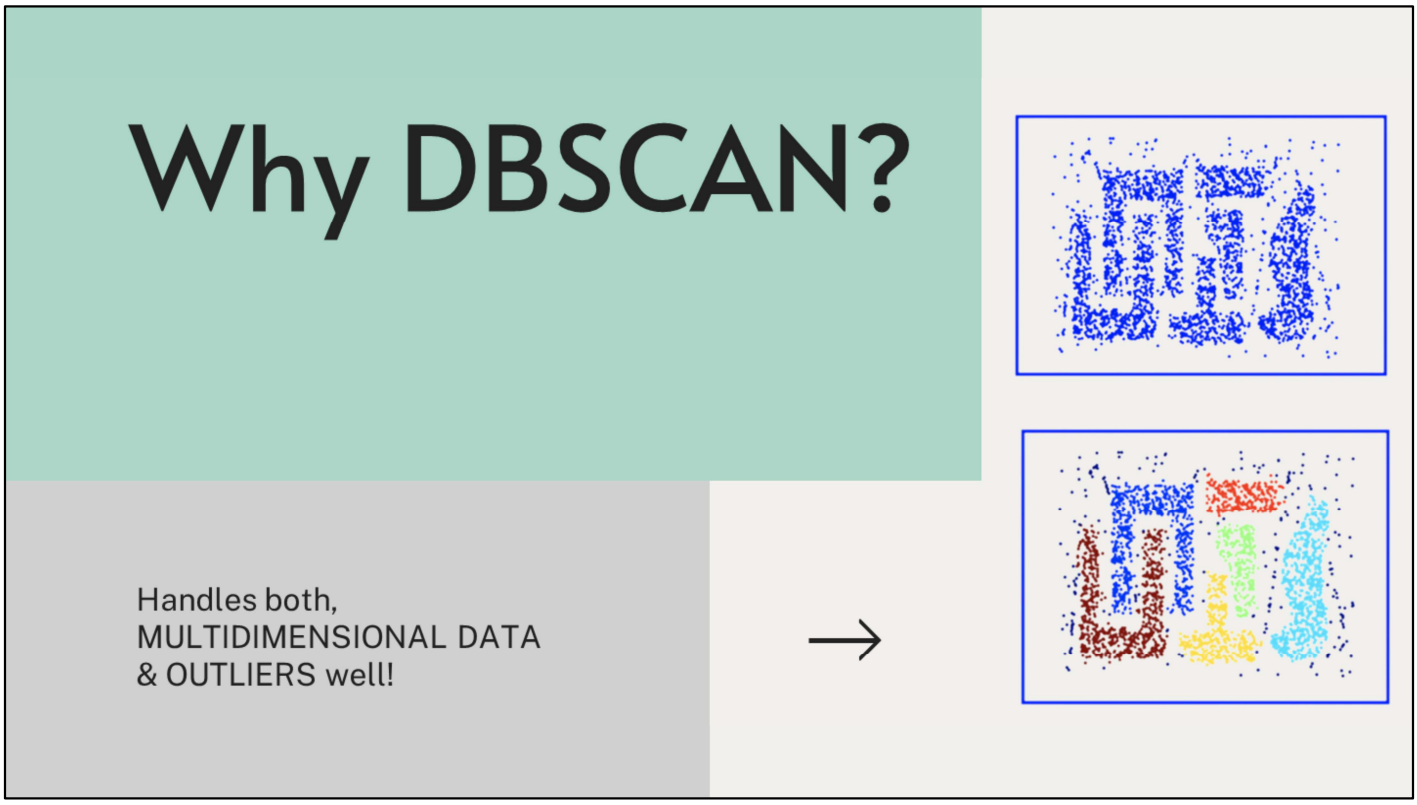


contd.

02 – DBSCAN

- After one cluster is found, DBSCAN will start the same process from another random point that has not been clustered into any group.
- The 'find and merge' process goes on until DBSCAN find another cluster and all the rest clusters.
- The rest data points that does not belong to any cluster are the outliers.





The top image depicts a more traditional clustering method, such as K-Means, that does not account for multi-dimensionality. Whereas the below image shows how DBSCAN can contort the data into different shapes and dimensions in order to find similar clusters. We also notice in the below image, that the points along the outer edge of the dataset are not classified, suggesting they are outliers amongst the data

Market Segmentation

"Market segmentation involves viewing a heterogeneous market as a number of smaller homogeneous markets in response to differing preferences, attributable to the desires of consumers for more precise satisfaction of their varying wants"
– Smith, 1956

To ensure customer satisfaction and optimal profit, customer segmentation allows us to study and understand behaviors of customers.

[5]:

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

Our Database

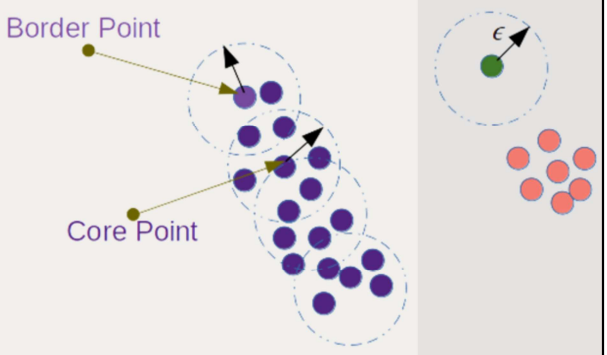
```
df = pd.read_csv('../input/Mall_Customers.csv')
df.head()
```

DBSCAN Inputs

DBSCAN works based on two parameters: Epsilon and Minimum Points

1. Epsilon determine a specified radius that if includes enough number of points within, we call it dense area.

2. minimumPoints determine the minimum number of data points we want in a neighborhood to define a cluster.



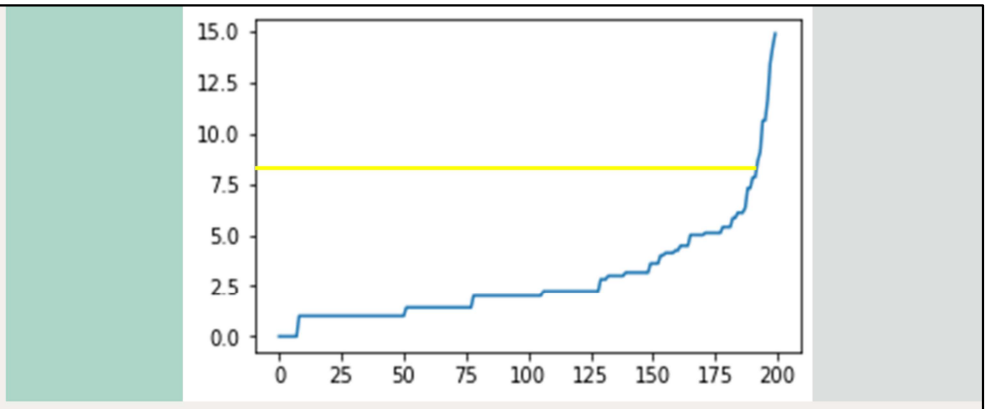
Also explain border points and noise/outliers

Getting minimumPoints

- The larger the data set, the larger the value of minPts should be, and if the data set is noisier, choose a larger value of minPts
- Generally, minPts should be greater than or equal to the dimensionality of the data set. (>= dim + 1)
- For 2-dimensional data, use DBSCAN's default value of minPts = 4 (Ester et al., 1996).
- If your data has more than 2 dimensions, choose minPts = 2*dim, where dim = the dimensions of your data set (Sander et al., 1998)

Here, dimensions = 5

Hence,
minPts = 2*5 = 10



Getting Epsilon

```
from sklearn.neighbors import NearestNeighbors
neighb = NearestNeighbors(n_neighbors = 9)
nbrs = neighb.fit(x)
distances,indices = nbrs.kneighbors(x)
```

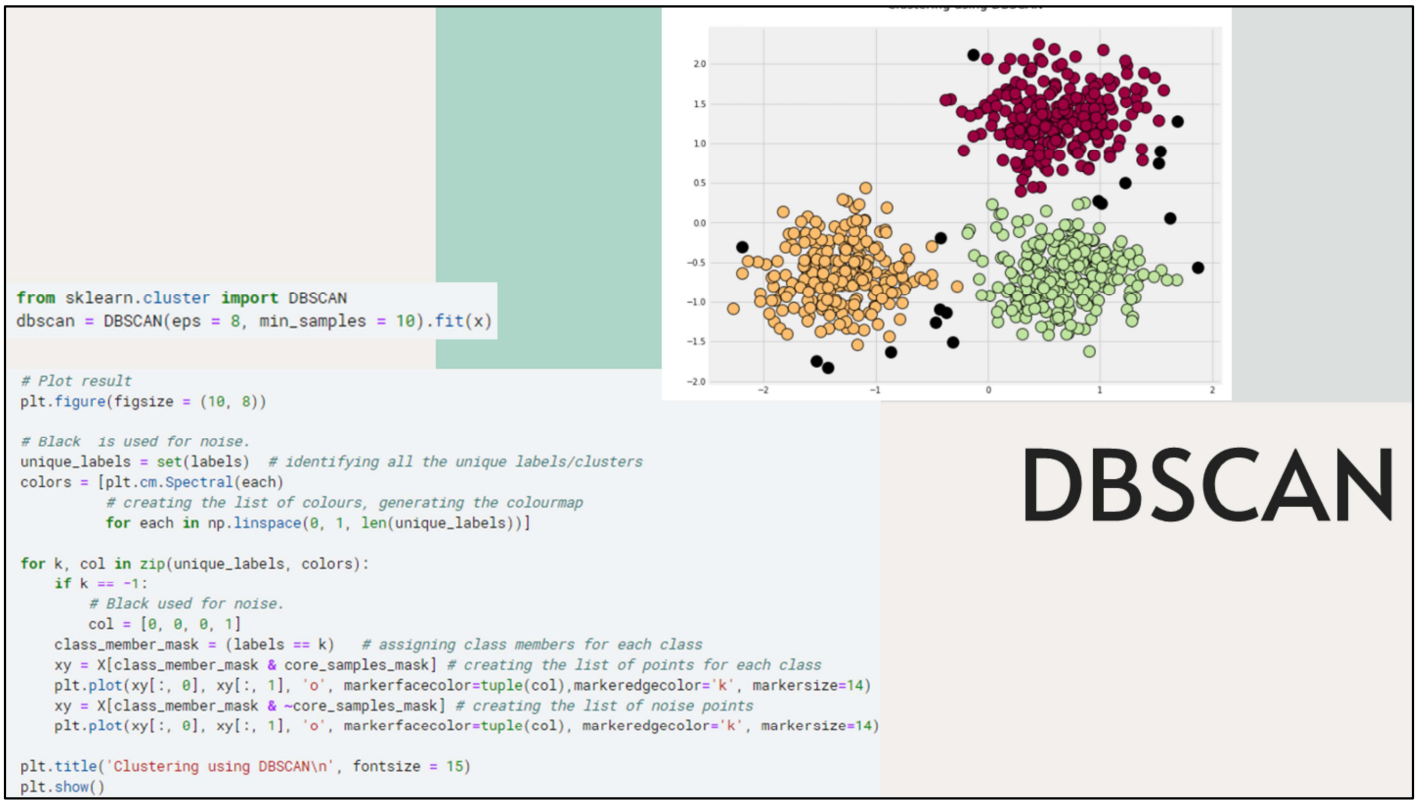
After you select your MinPts value, you can move on to determining ε. One technique to automatically determine the optimal ε value is described in a paper by Nadia Rahmah et. Al in 2012. This technique calculates the average distance between each point and its k nearest neighbors, where k = the MinPts value you selected. The average k-distances are then plotted in ascending order on a k-distance graph. You'll find the optimal value for ε at the point of maximum curvature (i.e. where the graph has the greatest slope).

eps: if the eps value chosen is too small, a large part of the data will not be clustered. It will be considered outliers because don't satisfy the number of points to create a dense region. On the other hand, if the value that was chosen is too high, clusters will merge and the majority of objects will be in the same cluster. The eps should be chosen based on the distance of the dataset (we can use a k-distance graph to find it), but in general small eps values are preferable.

Code -

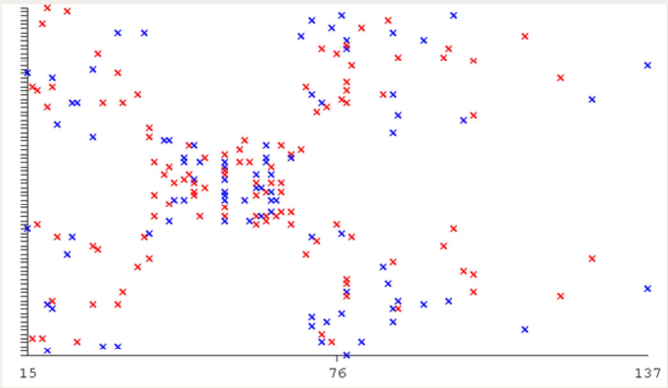
X is the values for the annual income and spending score columns since we're running our analysis on that.

After the written code, we sort the distances and plot this graph using matplotlib

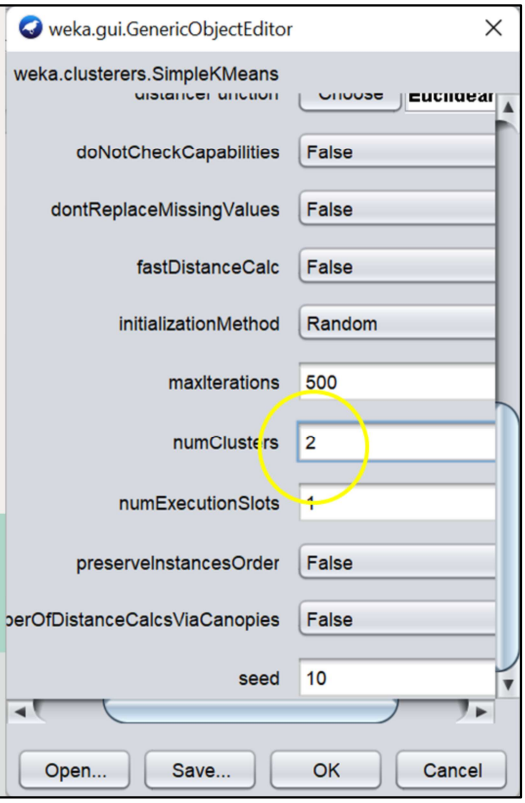

from sklearn.cluster import DBSCAN
dbscan = DBSCAN(eps = 8, min_samples = 10).fit(x)
# Plot result
plt.figure(figsize = (10, 8))
# Black is used for noise.
unique_labels = set(labels) # identifying all the unique labels/clusters
colors = [plt.cm.Spectral(each)
# creating the list of colours, generating the colourmap
for each in np.linspace(0, 1, len(unique_labels))]
for k, col in zip(unique_labels, colors):
if k == -1:
# Black used for noise.
col = [0, 0, 0, 1]
class_member_mask = (labels == k) # assigning class members for each class
xy = X[class_member_mask & core_samples_mask] # creating the list of points for each class
plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),markeredgecolor='k', markersize=14)
xy = X[class_member_mask & ~core_samples_mask] # creating the list of noise points
plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col), markeredgecolor='k', markersize=14)
plt.title('Clustering using DBSCAN\n', fontsize = 15)
plt.show()
DBSCAN

(Analysis)

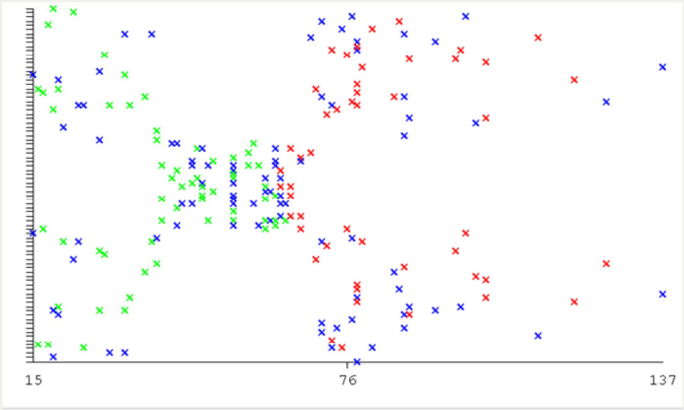
Number of Clusters = 2



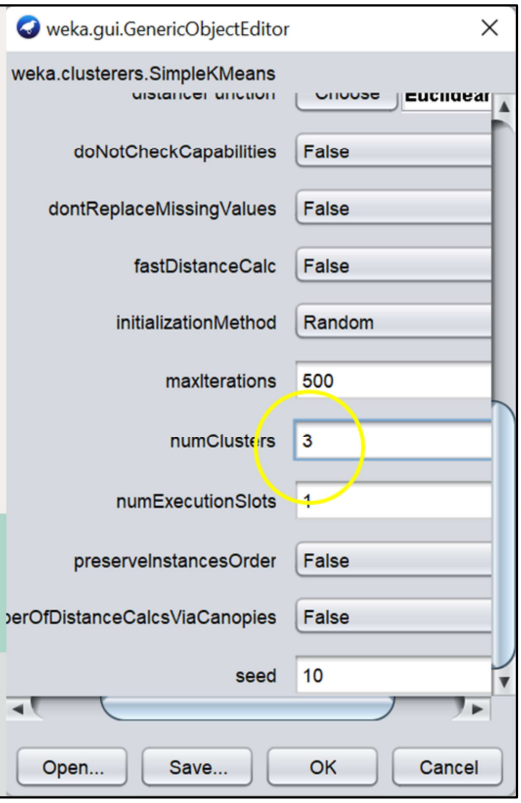
WEKA: Simple K-Means



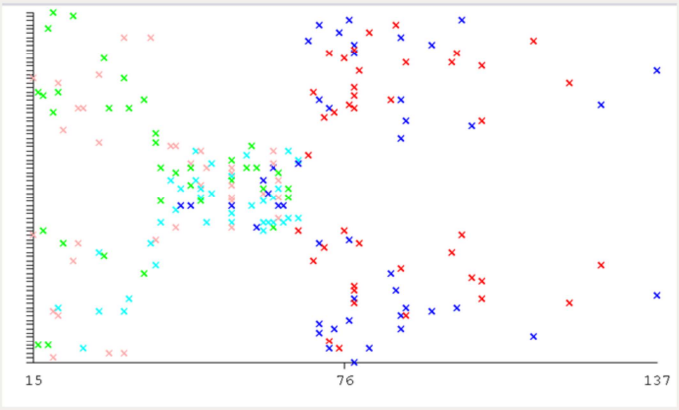
Number of Clusters = 3



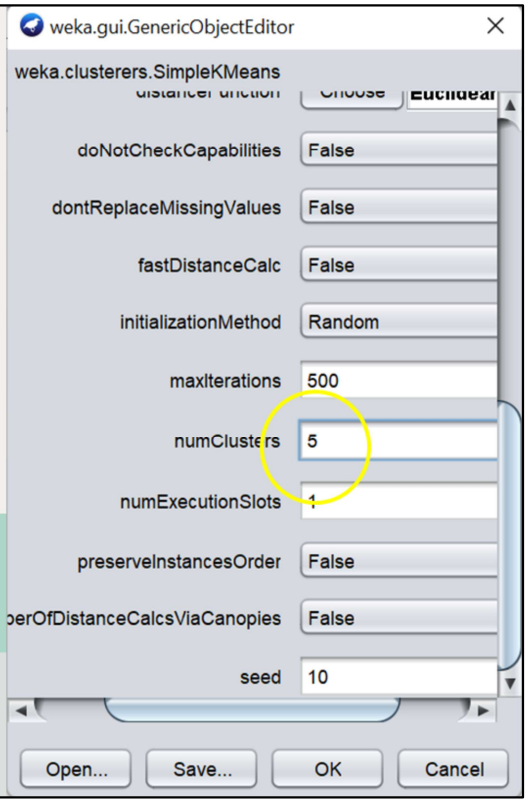
WEKA: Simple K-Means

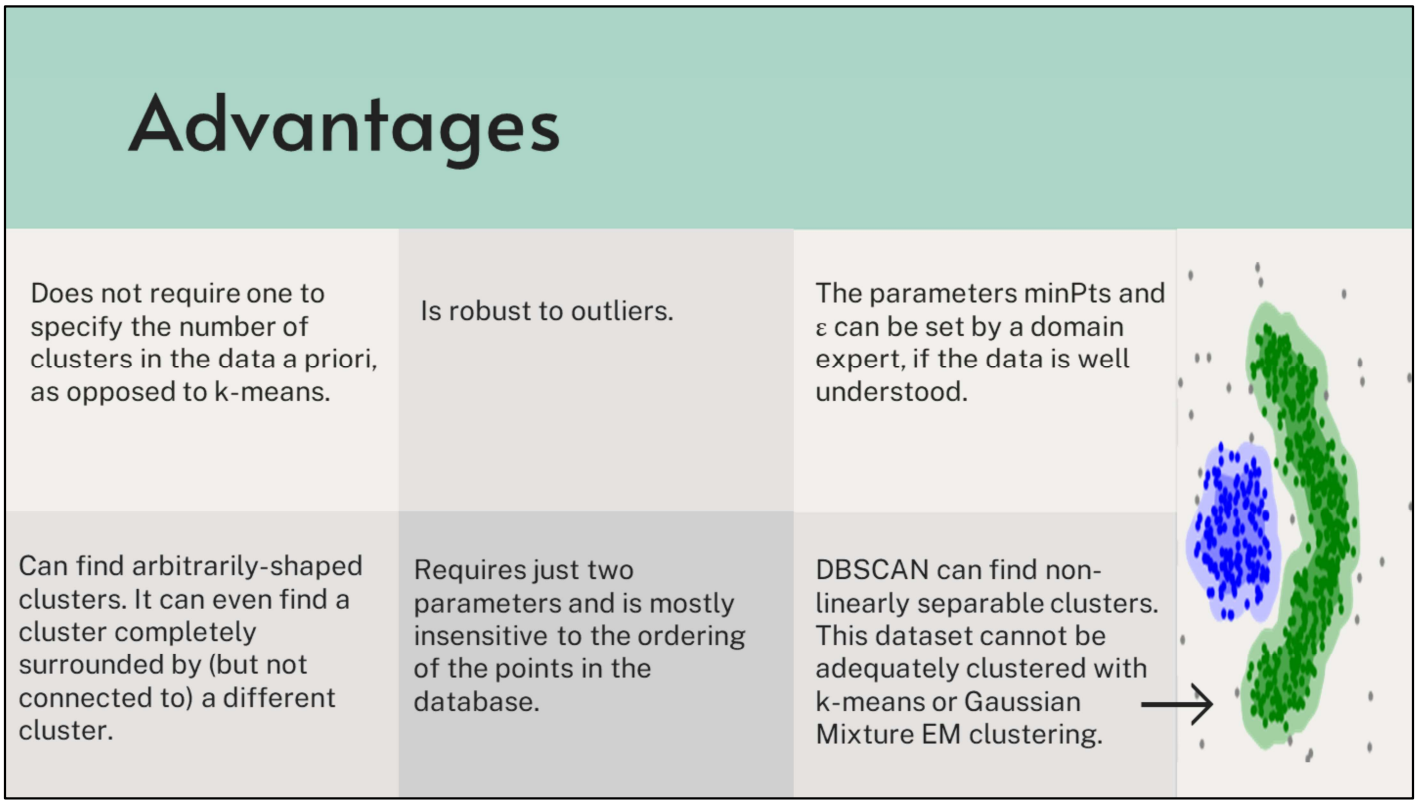


Number of Clusters = 5



WEKA: Simple K-Means



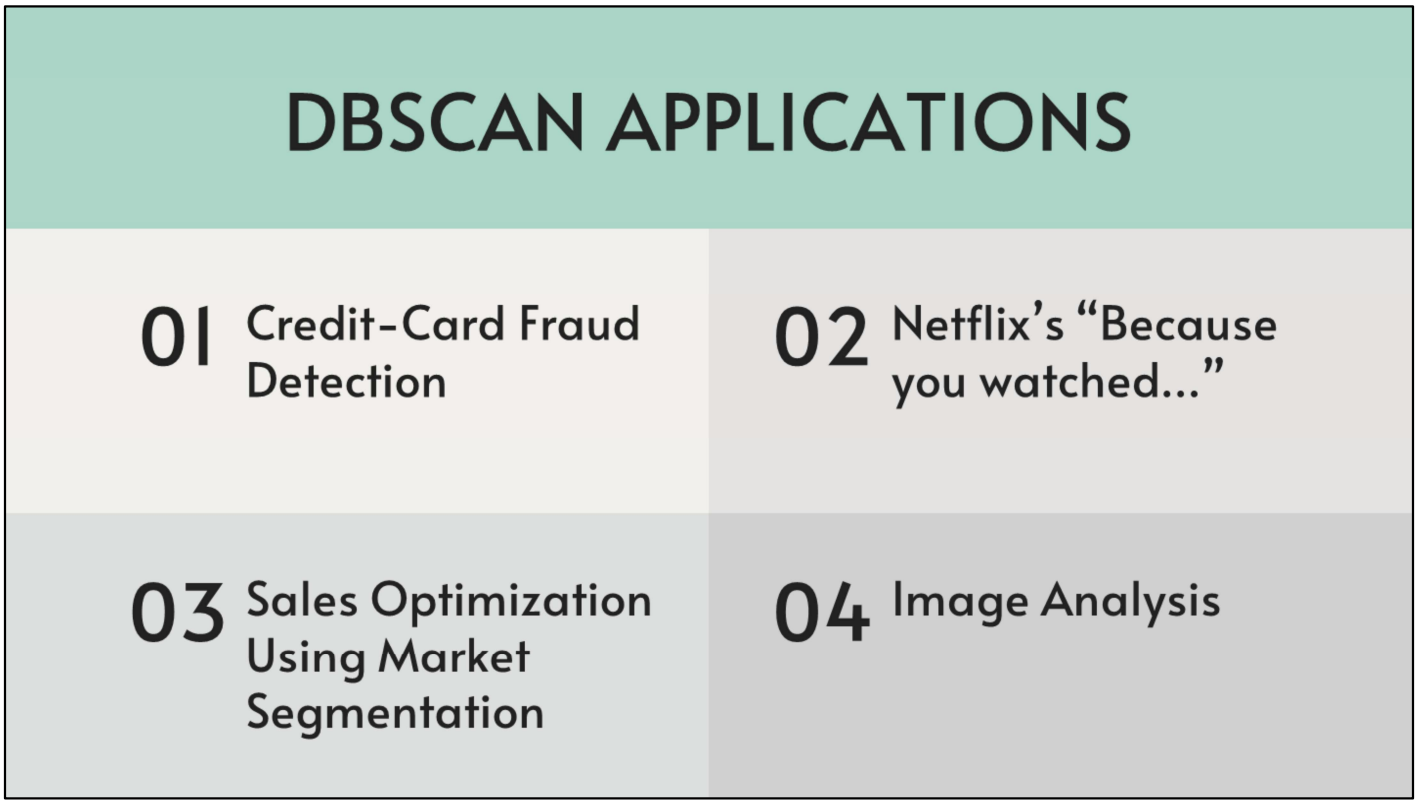


ADVANTAGES

1. DBSCAN does not require one to specify the number of clusters in the data a priori, as opposed to k-means.
2. DBSCAN can find arbitrarily-shaped clusters. It can even find a cluster completely surrounded by (but not connected to) a different cluster. Due to the MinPts parameter, the so-called single-link effect (different clusters being connected by a thin line of points) is reduced.
3. DBSCAN has a notion of noise, and is robust to outliers.
4. DBSCAN requires just two parameters and is mostly insensitive to the ordering of the points in the database. (However, points sitting on the edge of two different clusters might swap cluster membership if the ordering of the points is changed, and the cluster assignment is unique only up to isomorphism.)
5. DBSCAN is designed for use with databases that can accelerate region queries, e.g. using an R* tree.
6. The parameters minPts and ε can be set by a domain expert, if the data is well understood.

DISADVANTAGES –

1. DBSCAN is not entirely deterministic: border points that are reachable from more than one cluster can be part of either cluster, depending on the order the data are processed. For most data sets and domains, this situation does not arise often and has little impact on the clustering result:[4] both on core points and noise points, DBSCAN is deterministic. DBSCAN*[8] is a variation that treats border points as noise, and this way achieves a fully deterministic result as well as a more consistent statistical interpretation of density-connected components.
2. The quality of DBSCAN depends on the distance measure used in the function regionQuery(P,ε). The most common distance metric used is Euclidean distance. Especially for high-dimensional data, this metric can be rendered almost useless due to the so-called "Curse of dimensionality", making it difficult to find an appropriate value for ε. This effect, however, is also present in any other algorithm based on Euclidean distance.
3. DBSCAN cannot cluster data sets well with large differences in densities, since the minPts-ε combination cannot then be chosen appropriately for all clusters.[9]
4. If the data and scale are not well understood, choosing a meaningful distance threshold ε can be difficult.



Based on previous shows you have watched in the past, Netflix will recommend shows for you to watch next. Anyone who has ever watched or been on Netflix has seen the screen below with recommendations
Considering it is trying to predict the future with what show I am going to watch next, Netflix has nothing to base the predictions or recommendations on (no clear definitive objective). Instead, Netflix looks at other users who have also watched 'Shameless' in the past, and looks at what those users watched in addition to 'Shameless'. By doing so, Netflix is clustering its users together based on similarity of interests. This is exactly how unsupervised learning works. Simply clustering observations together based on similarity, hoping to make accurate conclusions based on the clusters.

Suppose we have an e-commerce and we want to improve our sales by recommending relevant products to our customers. We don't know exactly what our customers are looking for but based on a data set we can predict and recommend a relevant product to a specific customer. We can apply the DBSCAN to our data set (based on the e-commerce database) and find clusters based on the products that the users have bought. Using this clusters we can find similarities between customers, for example, the customer A have bought 1 pen, 1 book and 1 scissors and the customer B have bought 1 book and 1 scissors, then we can recommend 1 pen to the customer B. This is just a little example of use of DBSCAN, but it can be used in a lot of applications in several areas.

RESOURCES

Determination of Optimal Epsilon (Eps) Value on DBSCAN Algorithm to Clustering Data on Peatland Hotspots in Sumatra
by Nadia Rahmah, et. al.
DOI: doi:10.1088/1755-1315/31/1/012012

Density-Based Clustering,
by Joerg Sander, et. al.
DOI: 10.1007/978-0-387-30164-8_211

Customer Segmentation using Centroid Based and Density Based Clustering Algorithms,
by A. S. M. Shahadat Hossain, et. al.
DOI: 10.1109/EICT.2017.8275249

https://jovian.ai/manishagdas/customer-segmentation-using-dbscan

RESOURCES

https://elutins.medium.com/dbscan-what-is-it-when-to-use-it-how-to-use-it-8bd506293818

https://www.kaggle.com/code/niteshyadav3103/customer-segmentation-using-kmeans-hc-dbscan/

Spyder Ananconda for Python 3.9

Google Colaboratory

Kaggle Notebooks

Vodka + Caffeine ☺