Hierarchical Clustering

R I B I E S DATE:

- Hierarchical clustering is very common in biology and lend itself nicely to visualizing clusters.
- It can also help the user decide on an appropriate number of clusters.

Section Overview:-
- Theory and Intuition of Hierachical Clustering
- Coding Example of Hierarchical clustering
- .

Hierarchical clustering

Theory and Intuition

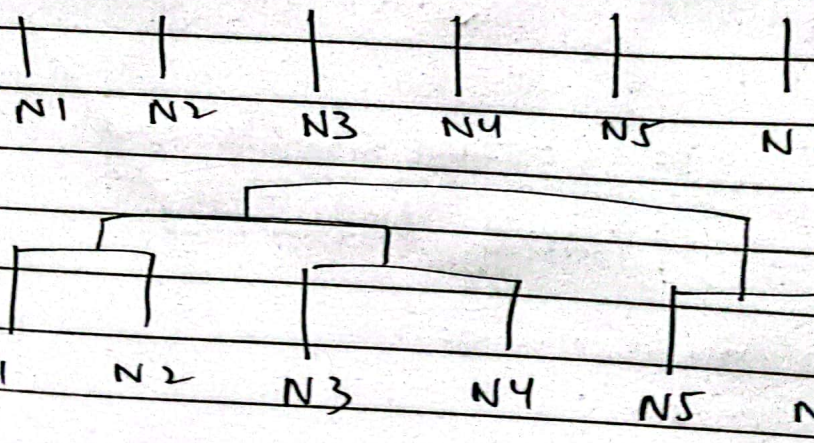
- Like most clustering algorithms, Hierarchical clustering simply relies on measuring which data point are most "similar" to other data points.
- "Similarity" is defined by choosing a distance metric
- So why use Hierarchical clustering?
  - Easy to understant and visualize
  - Help users decide how many clusters to choose.
  - Not necessary to choose clusters. amount before running the algorithm

DATE:

M T W T F S

- Divide points into potential clusters:
- Agglomerative Approched:
  - Each point begins as its own clusters, then clusters are joined.
- Divisive Approach:
  - All points begin in the same cluster, then clusters are split.

- Agglomerative



- Opposite of the Agglomeractive approch is a Divisive approach, which start with all point belonging to the same cluster, and the begins division to seperate out clusters.

NOTES DATE:

- Hierarchical Clustering process:
  - Compare data points to find similar datapoint to each other.
  - Merge these to create a cluster.
  - Compare clusters to find most similar clusters and merge again
  - Repeat untill all points in a single cluster.

There are a few key topics we still need to understand for Hierarchical clustering:
- Similarity Metric
- Dendogram
- Linkage matrix

- Similarity metrics
  - Measure distance between two points
  - many options
    - Euclidean Distance
    - Manhattan
    - cosine
    - and many more
  - Default choice is Euclidean

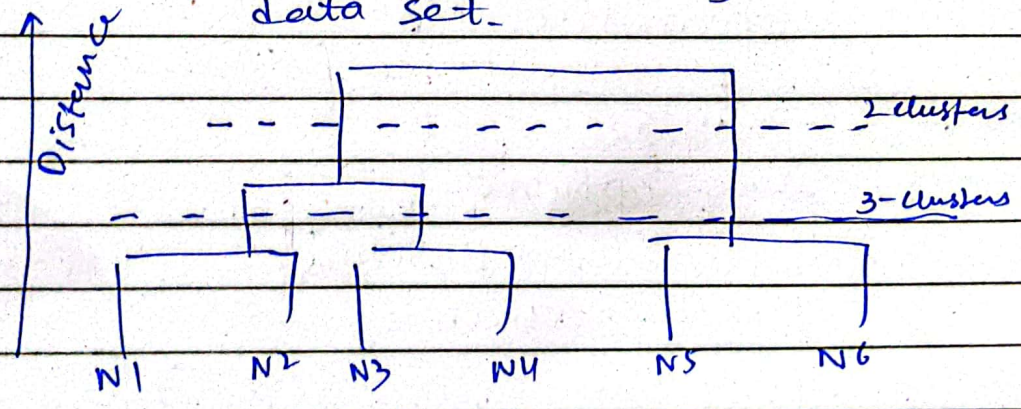
$$d(p,q)^2 = (q_1-p_1)^2 + (q_2-p_2)^2$$

M T W T F S DATE:_____

- Each dimension would be a features
- for n data points and p features:
  - $D^2 = (X_{11} - X_{12})^2 + \cdots + (X_{n-1p-1} - X_{np})^2$
- using MinMax Scaler we can scale all features to be between 0 and 1.
- This allow for maximum distance between a features to be 1.

Dendrogram:-

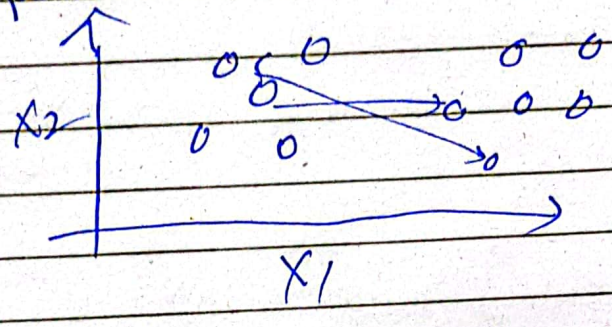
- plot displaying all potential clustering.
- very computationally expensive to compute and display for larger data set.



- linkage:-
  - How we measure distance from a point to an entire cluster?
  - How do we measure distance from a cluster to another cluster?

MIDIDES DATE:

- Once two or more points are together and we want to continue agglomerative clustering to join cluster we need to decide on a linkage parameter-



- criterion determining which distance to use between sets of observation-
- Algorithm will merge pair of clusters that minimizes the criterion.

Ward: minimize variance of clusters being merged-

Average: use average distance between two sets-

- Minimum or maximum distances between all observation of the two sets-

003 Hierical clustering coding # part 2

M T W T F S DATE:_____

```
df=pd.read_csv('cluster_mpg.csv')
df
df.describe()  ()


df['origin'].value_counts()
df_w_dumies = pd.get_dumies(df.drop("have", axis=1)
df_w_dumies

from sklearn.preprocessy MinMaxScaler

# sqrt
  scaler = MimaxScaler()
scaled_data = scaler.fit_transform(df_w_dumies)
scaled_data

scaled_df = pd.DataFrame(scaled Data, column=df_w_dumies)
scaled_df

plt.figure(figsize=(5,8))
sns.heatmap(scaled_df, cmap=' ')

sns.clustermap(scaled_df, row_cluster=False
(scaled_df, col_cluster=False)
col map
```

- Hierarchical clustering

coding # part two

DATE:

Cluster and Dendrogram

```
# code

from sklearn.cluster import
                    Agglomerative
                    Clustering

model = AgglomerativeClustering(n_cluster=4)

cluster_labels = model.fit_predict(scaled_df)

plt.figure(figsize
sns.scatterplot(data=df, x='mpg', y='weight',
                hue =
                cluster
                labels
                palette='viridis')

hyperparameter
affinity: "euclidean"

#max distance possible?
# sqrt(N-features)

len(scaled_df.columns)

np.sqrt(10) # max distance possible
            between two points
            (min/Max scaling)

model = AgglomerativeClustering(n_clusters=
                    None,
                    distance_threshold=0)
```

M T W T F S DATE:

```
cluster_labels = model.fit_predict
                    (scaled_df)
```

then ~~[illegible]~~ -

```
from scipy.cluster.hierarchy import
                    dendrogram
from scipy.cluster import hierarchy
```

```
linkage matrix = hierarchy.linkage(
                    model.children_)
```

```
plt.figure(figsize=(20,10),)
dendro = ~~hdbscan~~ dendogram(linkage matrix)
    truncate_mode = 'lastp', p=10)
                    'level', p=3
```

```
np.sqrt(len(scaled_df.columns))
```

```
scaled_df['mpg'].idxmin()
```

```
car_a = scaled_df.iloc[350]
car_b = scaled_df.iloc[28]
```

M T W T F S DATE:_____

distance = np.linalg.norm(car_a - car_b)

distance (2.3852)

DBSCAN

M T W T F S DATE:_____

- DBSCAN - Density-based spatial clustering of application with noise is a powerful technique which can be used for clustering and outliers detection.
- lets review what this section will cover

- Section Overview:-
  - DBSCAN vs. KMEAN clustering
  - DBSCAN Hyperparameters Theory
  - BBscan Hyperparamters coding
  - Outlier project Exercise
  - Project solutions.

Theory and Intuition

DBSCAN = Density-based spatial clustering of application with noise

- 1972: Robert F. Ling published a closely related algorithm in "The theory and construction of k-clusters" with an expected run time of $O(n^3)$.
- This mean that as n number of points grows, the run time of

MUDIES DATE:

the algorithm grows cubically!

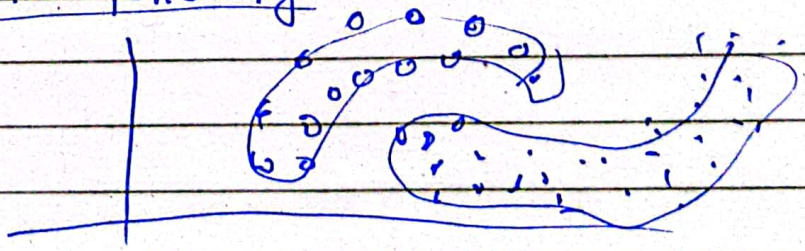
- 1996: Martin Ester, Hans-peter Jorregel, Jörg sander and Xiowei Xu proposed the modern version of DBSCAN with a runtime of $O(n^2)$.

- 2014: DBSCAN was awarded the test of time award at the leading data mining conference, SIGKDD.

Question
- How its work?
- Advantages and disadvantages
- How it deal outlier and noise

DBSCAN Key Idea:
- DBSCAN focusses on using density of points as its main factor for assigning cluster label.
- This create the ability to find cluster segmentations that other algorithms have difficulty with.

Consider the following:-



M T W T F S DATE:

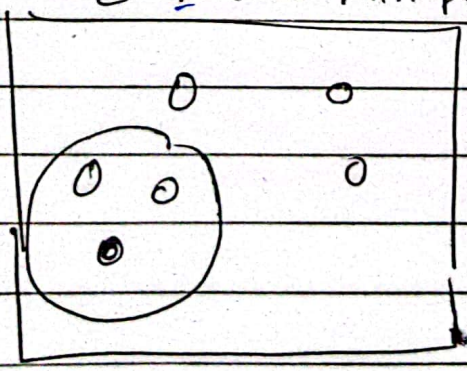
- DBSCAN iterates through points and uses two key ~~value~~ hyperparameters (epsilon and minimum number of points) to assign clusters labels.
- unlike kmean, it focuses on density as the main factor for clustering assignment of points.

- DBSCAN key hyperparameter:
  - Epsilon
    - Distance extended from a point.
  - Minimum number of points
    - minimum number of points in an epsilon distance.

- DBSCAN point types:
  - Core
  - Border
  - Outlier

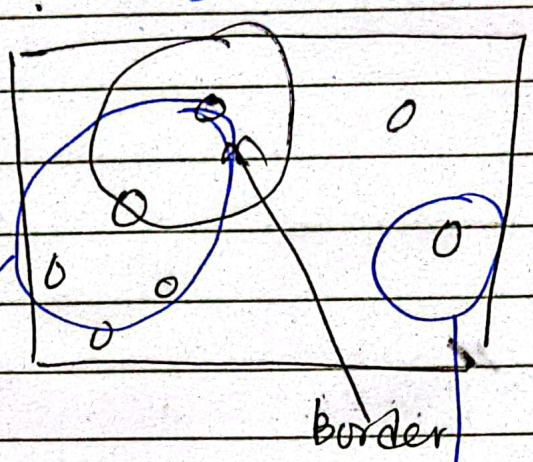
$\varepsilon = 1$ and Min points = 2

Core:
points with min points in epsilon range. (include itself)



M T W T F S DATE:_____

Border: $\varepsilon = 1$ and Min points = 3



- In epsilon range of core point, but does not contain min. number of points.

Outlier:-
Can not be "reached" by points in a cluster assignment.

- DBSCAN procedure:
  - pick a random point not yet assigned.
  - Determine the point type
  - Once a core point has been found, add all directly reachable points to the same cluster as core.
  - Repeat untill all points have been assigned to a cluster or an outlier.

coding

M T W T F S

DATE:

```
blobs = pd.read_csv('.../Data cluster blobs')
blobs.head()

sns.scatterplot(data=blobs, x="X1",
                y='X2')

moons = pd.read_csv('.../Data/cluster_
                    moons.csv')
sns.scatterplot(data=moons, x='X1', y='X2')

circles = pd.read_csv('.../.../cluster_circle.
                      csv')
circle.head()

def display_category(model, data):
    labels = model.fit_predict(data)
    sns.scatterplot(data=data, x='X1',
                    y='X2',
                    hue=Labels,
from sklearn.clusters =3 imperpalette='Set1

disp model = KMeans(n_clusters=2)
    display_categorical(model, moons)

from sklearn.cluster import DBSCAN
model = DBSCAN()
display_categories(model, blobs)
```

MIDIES DATE:

```
model = DBSCAN(eps=0.15)
display=categories(model, moons)
```

Hypermeter theory :-

Epsilon:
- Distance extended from a point to search for Min. Number of points.

Min. Number of points:
- Minimum number of points within Epsilon distance to be a core point

Adjusting these hyperparameter have two main outcome:
- changing number of clusters.
- changing what is an outlier point.

- Epsilon Intuition:-
  - Increasing epsilon allows more points to be core points which also result in more border points and less outlier points
  - Imagine a huge epsilon, all points would be would be within the neighbourhood and classified as the same clusters!

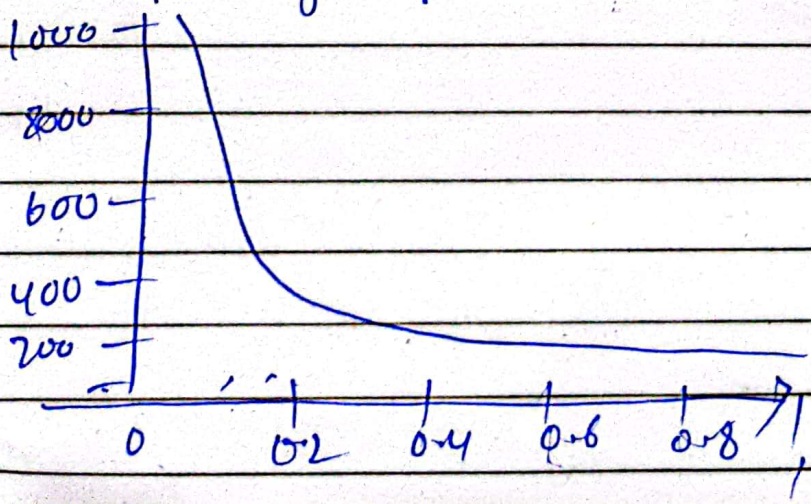
M T W T F S DATE:

- Decreasing epsilon causes more points not to be in range of each other, creating more unique clusters.
- Imagine a tiny epsilon, the range would ~~be~~ not extend far out enough to come into contact with any other points

Methods for finding an epsilon value:

- Extremely dependent on the particular data set and domain spaces.
- Requires user to have some expectation or intuition about number of clusters and relative percentage and relative percentage of outliers.

plot 'elbow/knee" diagram comparing epsilon value



MIDIES DATE:

- Minimum Number of Samples/points
  -~~Number of~~ 1
  - Increasing to a larger number of ~~po~~ samples needed to be considered a one point, causes more point to be considered unique outliers.
  - Imagine if min. number of sample was close to total number of points available, then very likely all points would become outliers.

Choosing Minimum number of Sample
- Test multiple potential values and chart againts number of outliers labeled.

Usefull to increase to create potential new small clusters, instead of complete outliers.

Coding

M T W T F S

DATE:

```
two_blobs = pd.read_csv("two_blobs")
two_blobs_outlier = pd.read_csv("")
sns.scatterplot(data=two_blobs,
                x='X1',
                y='X2')
```

sns. two_blobs_outlier

```
def display_categories(model, data):
    labels =
    //
    sns.scatterplot(data=
```

```
model = DBSCAN()
display_categorical(dbscan, two_blobs_o)
```

```
dbscan = dBScan(eps=0.001)
```
only outliers

```
dbscan = DBSCAN(eps=10)
```
one whole category

```
np.sum
(dbscan.labels == -1)
```

```
# Percent of outliers
100 * np.sum(dbscan.labels == -1)
                              / len
                              dbscan.la
```

outl

M T W T F S DATE:

```
outlier_percent = []
number_of_outliers = []

for eps in np.linspace(0.001, 1, 100):
```

// check jupyter notebook