Week 8: Clustering

M P Gururajan, Hina Gokhale and N N Viswanathan

Department of Metallurgical Engineering & Materials Science
Indian Institute of Technology Bombay

September, 2024

Outline

Clustering: example

ICI data

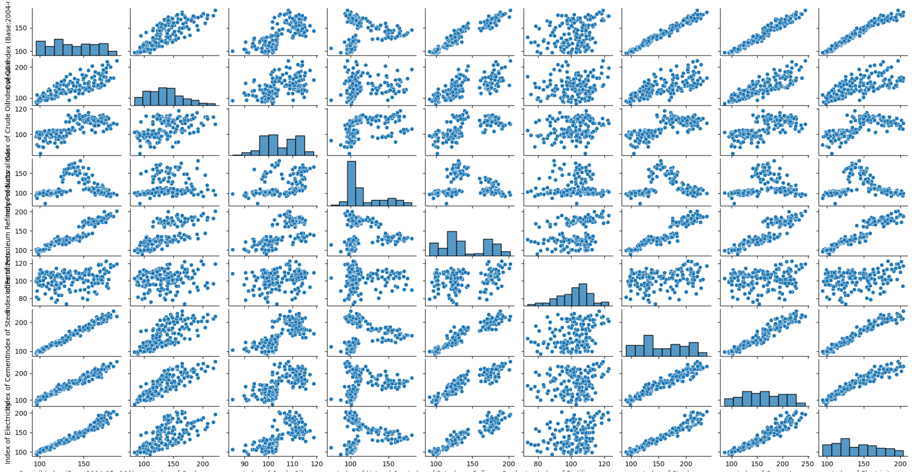
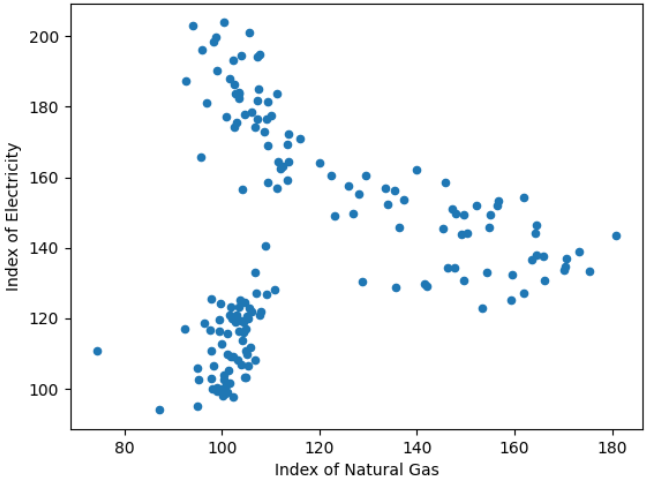
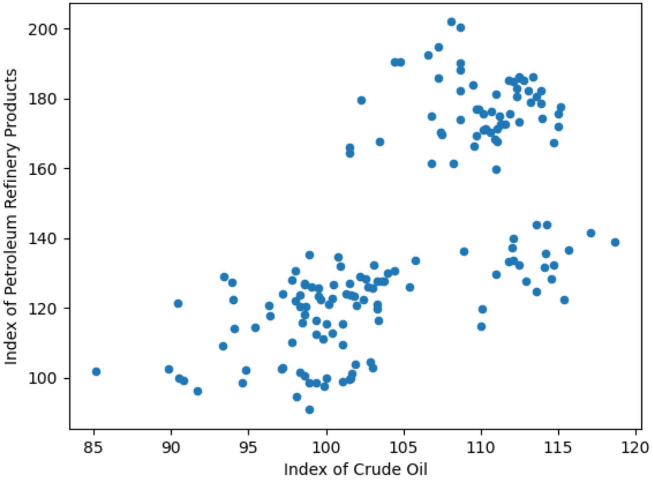


Figure: Plot using the `pairplot` command of the `seaborn` library.

A closer look



Another plot!



Automatic labelling

- Data: just the indices
- Response: not part of data
- Can the machine learn to cluster the data into appropriate groups?
- Yes!
- Clustering: an unsupervised learning algorithm

K-Means clustering

K-Means clustering

```
from sklearn.cluster import KMeans
k_means = KMeans(n_clusters = 2, init="k-means++",
max_iter=999,n_init=1,random_state=101)
k_means.fit(TestData)
plt.scatter(TestData['Index of Natural Gas'],
TestData['Index of Electricity'],
c=k_means.labels_.astype(float))
plt.xlabel("Index of Natural Gas")
plt.ylabel("Index of Electricity")
plt.scatter(k_means.cluster_centers_[:,0],
k_means.cluster_centers_[:,1],
c='red',marker='s')
plt.show()
```

K-Means clusters

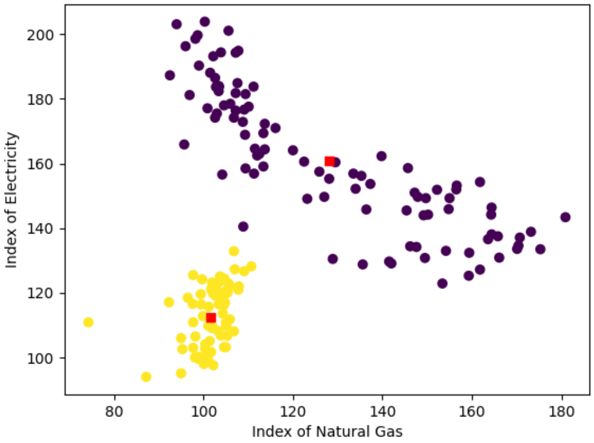


Figure: K-Means clustering algorithm separates the given data into two clusters. The centres of gravity of the clusters are marked by the red squares.

K-Means clusters

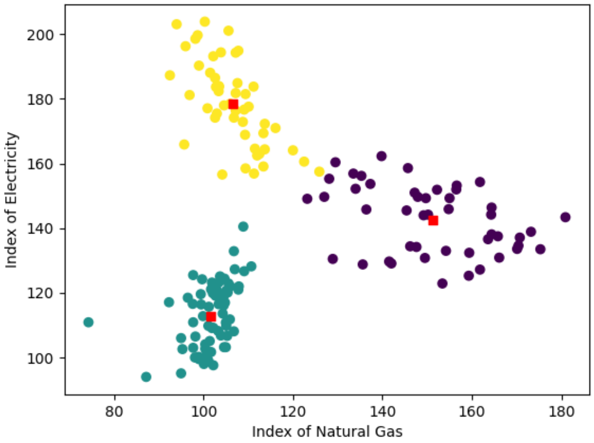


Figure: K-Means clustering algorithm separates the given data into three clusters. The centres of gravity of the clusters are marked by the red squares.

K-Means: 2 clusters

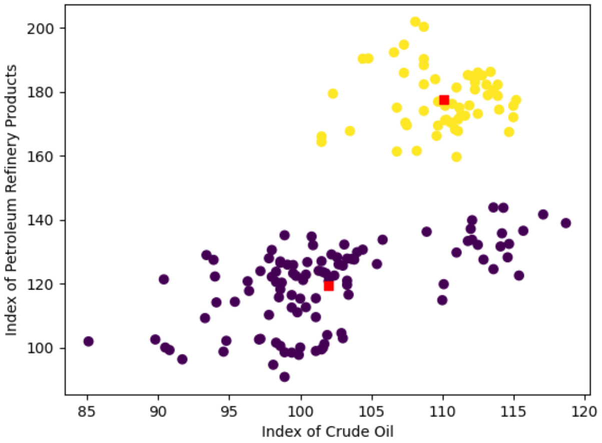


Figure: K-Means clustering algorithm separates the given data into two clusters. Result seems intuitively obvious.

K-Means: 3 clusters

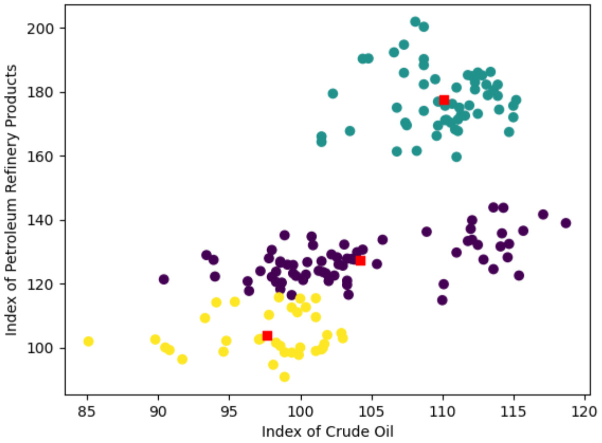


Figure: K-Means clustering algorithm separates the given data into three clusters. Result is not intuitively obvious.

K-Means: 4 clusters

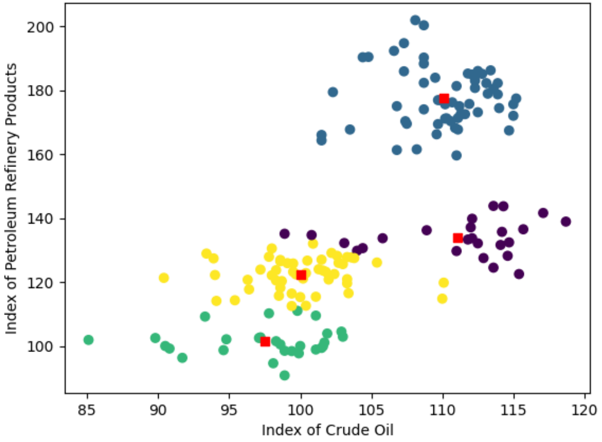


Figure: K-Means clustering algorithm separates the given data into four clusters.

K-Means: 5 clusters

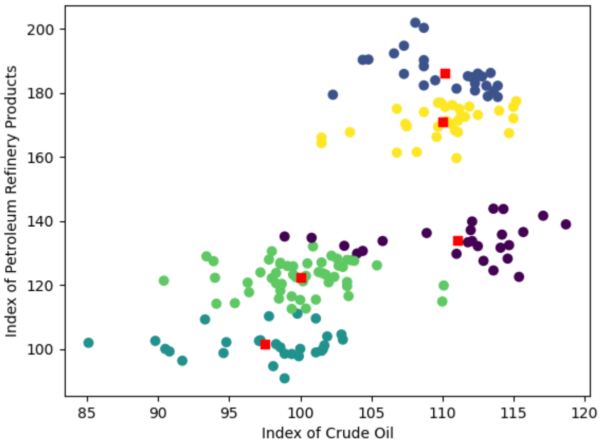


Figure: K-Means clustering algorithm separates the given data into five clusters.

K-Means: effect of initialisation

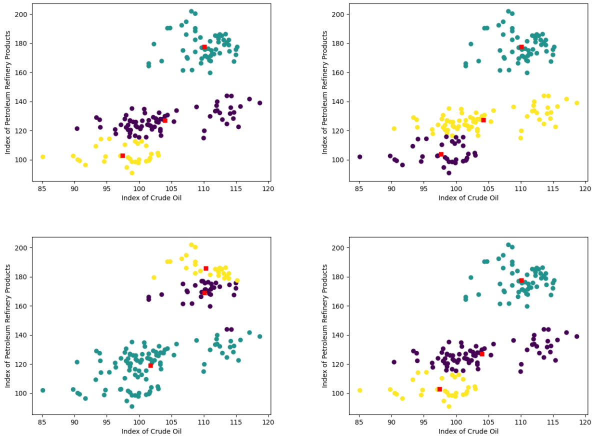


Figure: Effect of seeds 1, 11, (top) 1000 and 1001 (bottom).

Mini batch K-Means

```
from sklearn.cluster import MiniBatchKMeans
bk_means = MiniBatchKMeans(n_clusters = 3,
        init="k-means++",max_iter=999,
        batch_size=10, n_init=1, random_state=101)
bk_means.fit(TestData)
plt.scatter(TestData['Index of Crude Oil'],
        TestData['Index of Petroleum Refinery Products'],
        c=bk_means.labels_.astype(float))
plt.xlabel("Index of Crude Oil")
plt.ylabel("Index of Petroleum Refinery Products")
plt.scatter(bk_means.cluster_centers_[:,0],
        bk_means.cluster_centers_[:,1],c='red',marker='s')
plt.show()
```

Mini Batch K-Means: 3 clusters

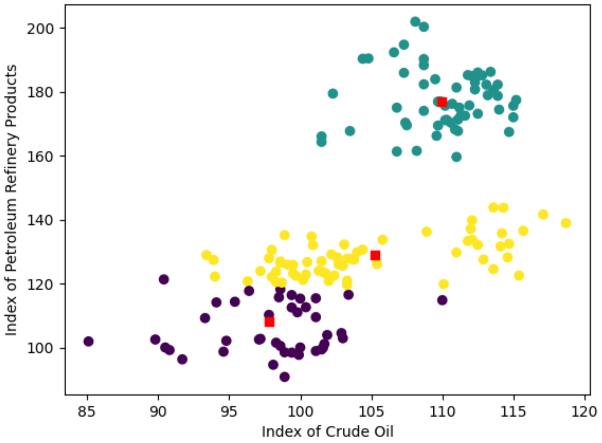
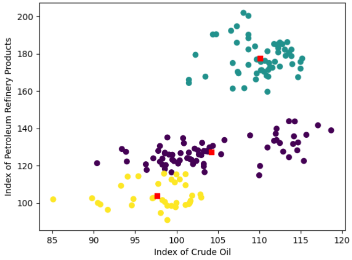


Figure: Mini batch K-Means clustering algorithm separates the given data into three clusters.

K-Means versus mini batch K-Means



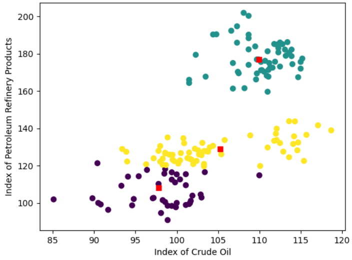


Figure: K-Means (left) and Mini batch K-Means (right) results.

A question

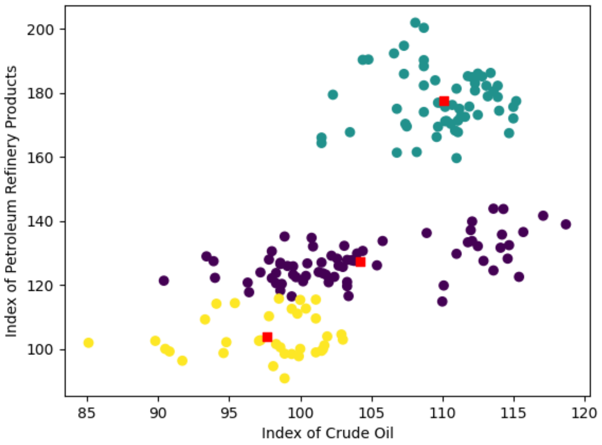


Figure: Why doesn't K-means give intuitively obvious clustering for this data?

Answer: data preprocessing is the key!

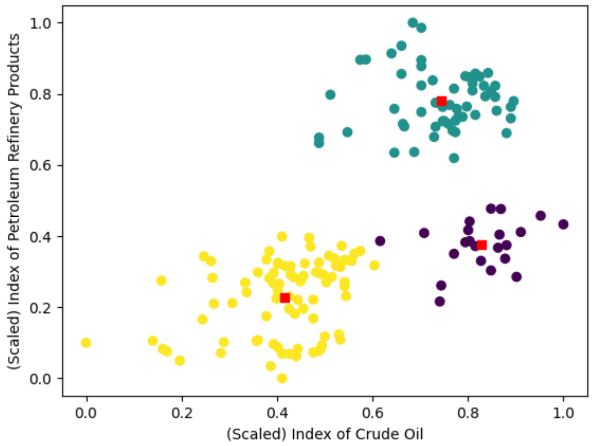


Figure: K-Means gives intuitively obvious results after scaling.

Scaling

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
ScaledData = scaler.fit_transform(TestData)
```

Why was scaling not needed in the first case?

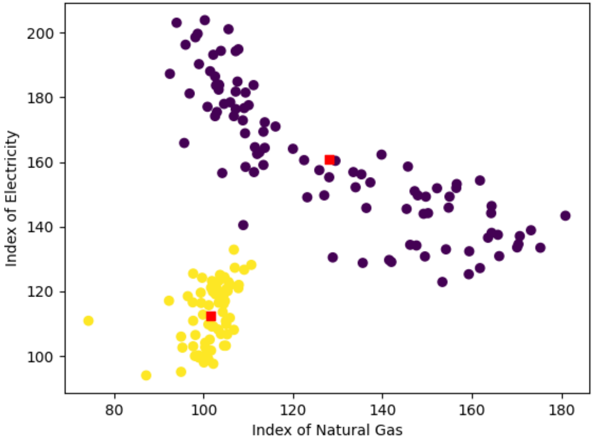


Figure: Notice that in this case the range was almost equal – accidentally!

K-Means clustering: algorithm

Notion of distance

- We know what distance means in 1-, 2- and 3-D
- Suppose we want to generalise this idea to N-dimensional space
- Distances are non-negative; it is zero when the starting and ending points are the same;
- Distance in symmetric; when you flip starting and ending you should get the same distance; and,
- Triangle inequality holds; there are no short cuts; if you have two distances between points A and C, and introduce a new point B, going from A to C via B is at best equal to the addition of these A to B and B to C; generally, it is greater than that.
- Examples: Euclidean distance (L_2 norm), Manhattan distance (L_1 norm), Chebyeshev distance (maximum of absolute difference between elements of vectors)

K-Means algorithm

- If there are k clusters, pick k random points as centroids of the clusters
- Depending on the Euclidean distance, all the points are assigned to the nearest centroid
- Recalculate the new centroids
- Reallocate points based on the new centroids
- Iterate till we find a stable solution
- Mini-batch version: split the data into chunks and process; useful for memory bound problems

Assumptions and tips: to remember

- Data consists of groups called clusters
- Groups are made of similar examples
- In the data space, groups have spherical shape!
- K-Means: assumes you know the number of clusters
- Good idea to standardize variables, and run K-Means after PCA
- Clustering is an unsupervised learning method: there is no measure of error
- You can calculate what is known as cohesion – how close are the data points in a given cluster – sum of square of distance from centroid – which is what was minimized in making the clusters!

Thank You!

Questions, clarifications, comments?