An Efficient K-means Clustering Algorithm

An analysis and Implementation

Contents:

1. Introduction to K-means Clustering Algorithm

- The algorithm partitions n observations into k clusters in which each observation belongs to the cluster whose center is nearest from it.

- Objective Function : Determine K centers so as To minimize the mean squared distance from each data point to its nearest center

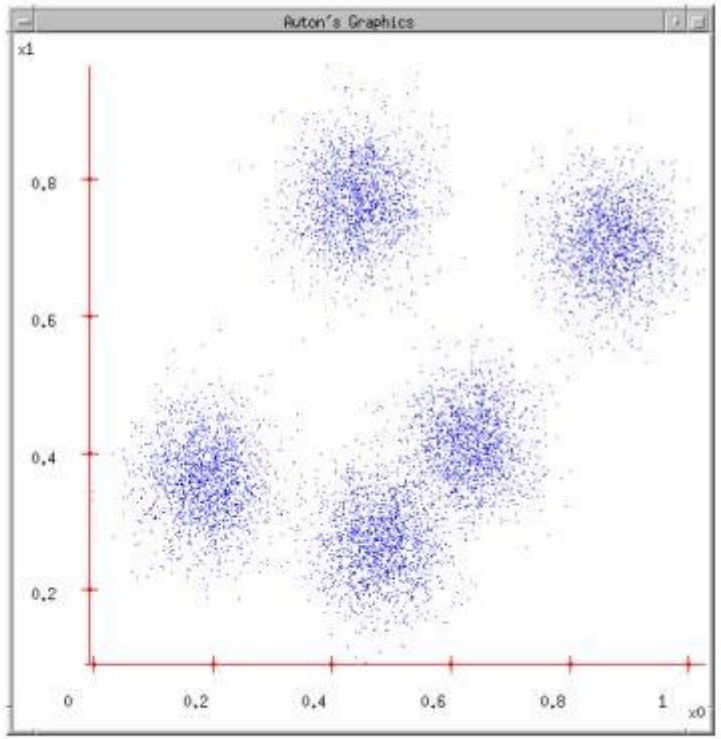
- Our Focus is on one of the popular variant of K-means clustering Algorithm called Llyod's Algorithm

- **Llyod's Algorithm**:
  1. Sample the centers at random from the data points
  2. For each stage of Llyod's Algorithm:
     - For each of the k centers, compute the centroid of the set of data points for which this center is closest
     - Move this center to the computed centroid and proceed to next stage.

MANCHESTER 1824

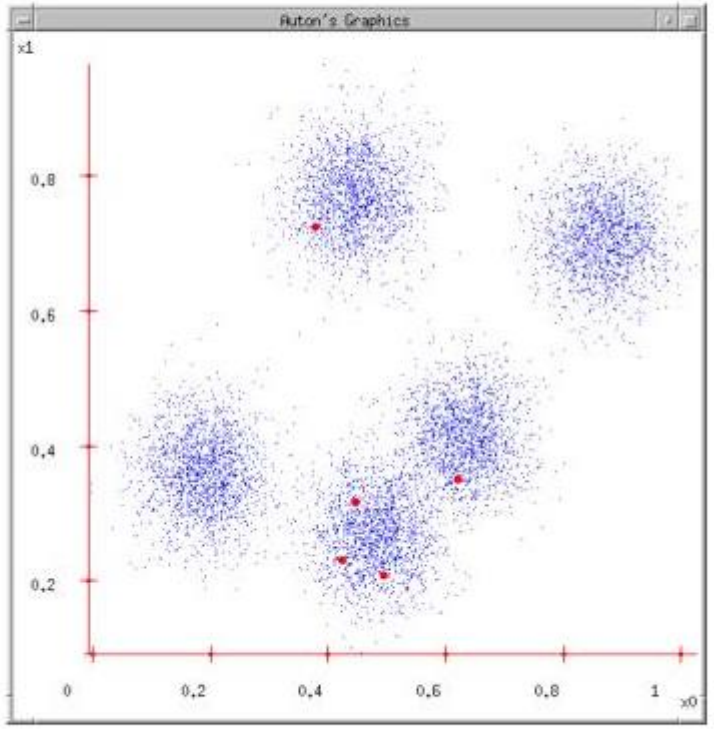
The University of Manchester

K-means Demo



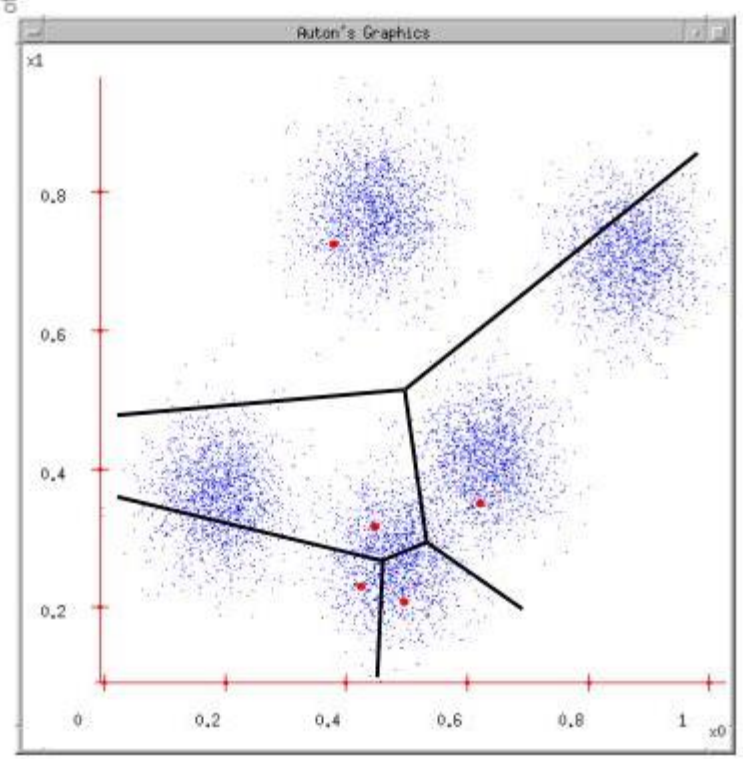
1. User set up the number of clusters they'd like. *(e.g. k=5)*

K-means Demo



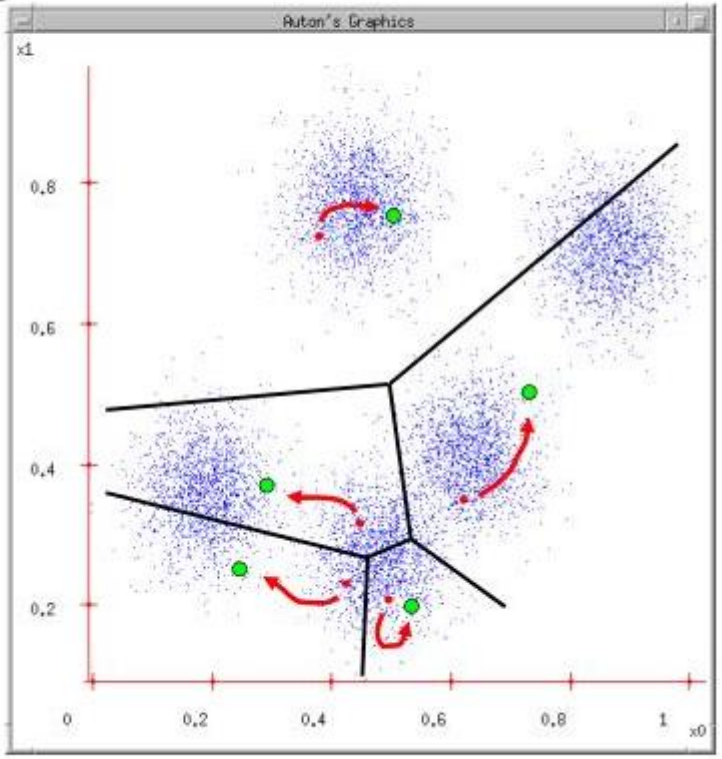
1. User set up the number of clusters they'd like. *(e.g. K=5)*
2. Randomly guess K cluster Center locations

K-means Demo



1. User set up the number of clusters they'd like. *(e.g. K=5)*
2. Randomly guess *K* cluster Center locations
3. Each data point finds out which Center it's closest to. (Thus each Center "owns" a set of data points)

K-means Demo

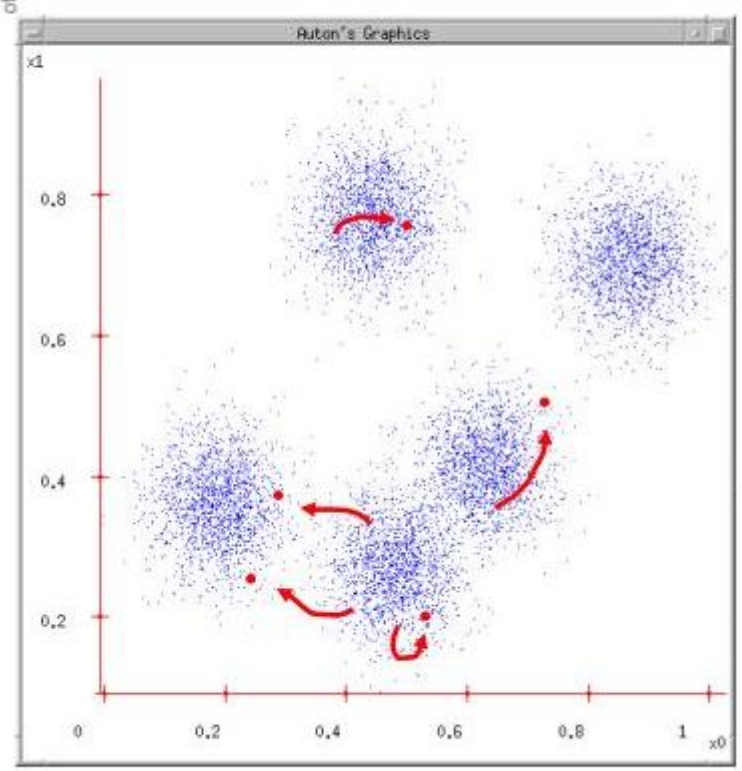


1. User set up the number of clusters they'd like. *(e.g. K=5)*
2. Randomly guess *K* cluster centre locations
3. Each data point finds out which centre it's closest to. (Thus each Center "owns" a set of data points)
4. Each centre finds the centroid of the points it owns

MANCHESTER 1824

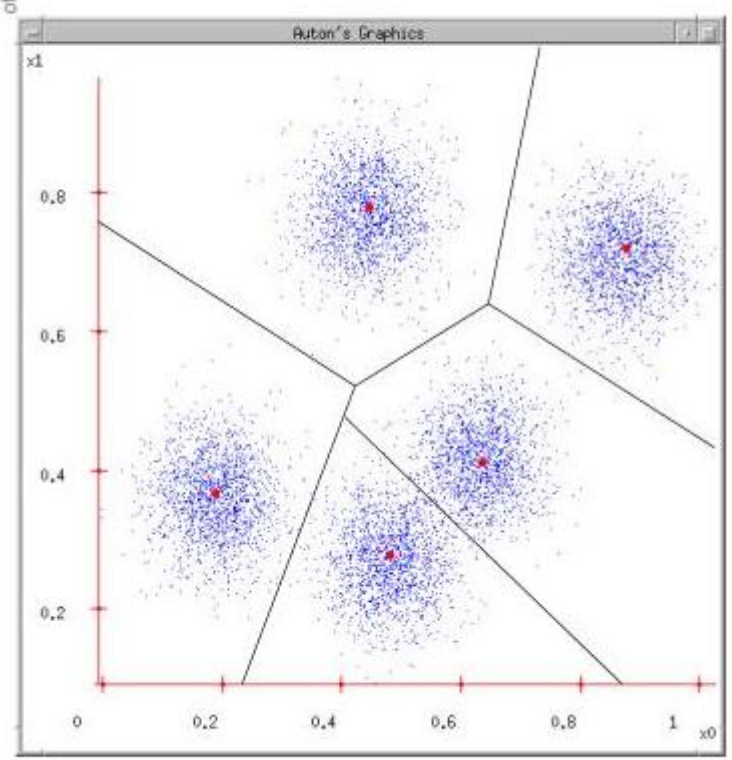
The University of Manchester

K-means Demo



1. User set up the number of clusters they'd like. (e.g. *K=5*)
2. Randomly guess *K* cluster centre locations
3. Each data point finds out which centre it's closest to. (Thus each centre "owns" a set of data points)
4. Each centre finds the centroid of the points it owns
5. …and jumps there

K-means Demo



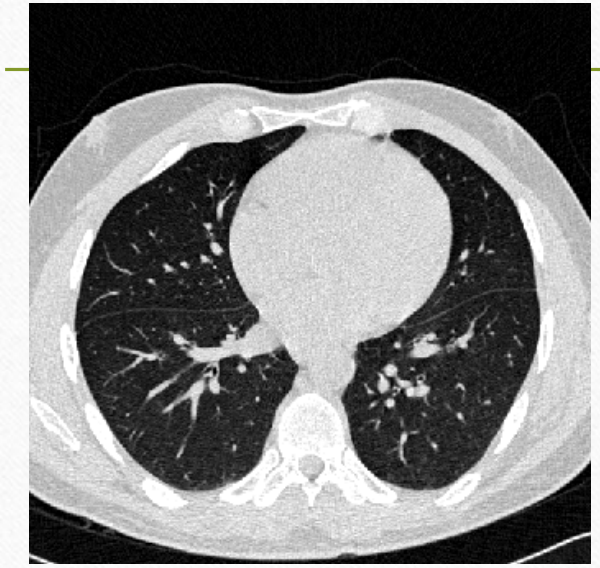
1. User set up the number of clusters they'd like. (e.g. *K=5*)
2. Randomly guess *K* cluster centre locations
3. Each data point finds out which centre it's closest to. (Thus each centre "owns" a set of data points)
4. Each centre finds the centroid of the points it owns
5. …and jumps there
6. …Repeat until terminated!

2. Importance of Work in Pattern Analysis and Machine Intelligence

2.1 Applications of k-means clustering:

- Image segmentation
- Color Quantization
- Recommendation Engines
- Fraud Detection
- Earthquake studies
- Biology

Image Segmentation Results



An image (*I*)

Three-cluster image (*J*) on gray values of *I*

2. Importance of Work in Pattern Analysis and Machine Intelligence

2.2 Problems with Existing variants of Llyod's algorithm:

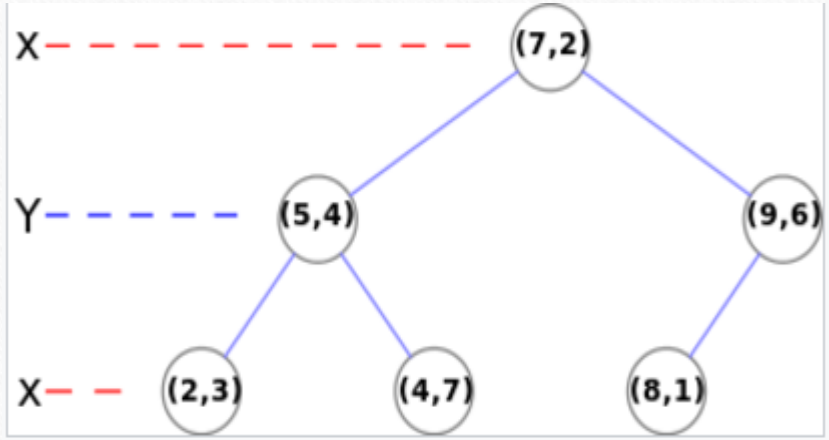
- Straightforward Implementation of Llyod's algorithm can be quite slow.
- This is due to the cost of computing the nearest neighbors from the data points
- Real life Applications have many more data points and higher k value. Hence clustering algorithms should be scalable both in terms of n and k
- But the Existing variants are not decently scalable if the **value of k or n increases**
  - **Brute Force Approach**: Computes the distance from every data point to every center
  - **Kd-center Algorithm** : operates by building kd-tree with respect to the center points
- Hence this paper provides an efficient implementation of Lloyd's algorithm called **Filtering/Pruning** Algorithm

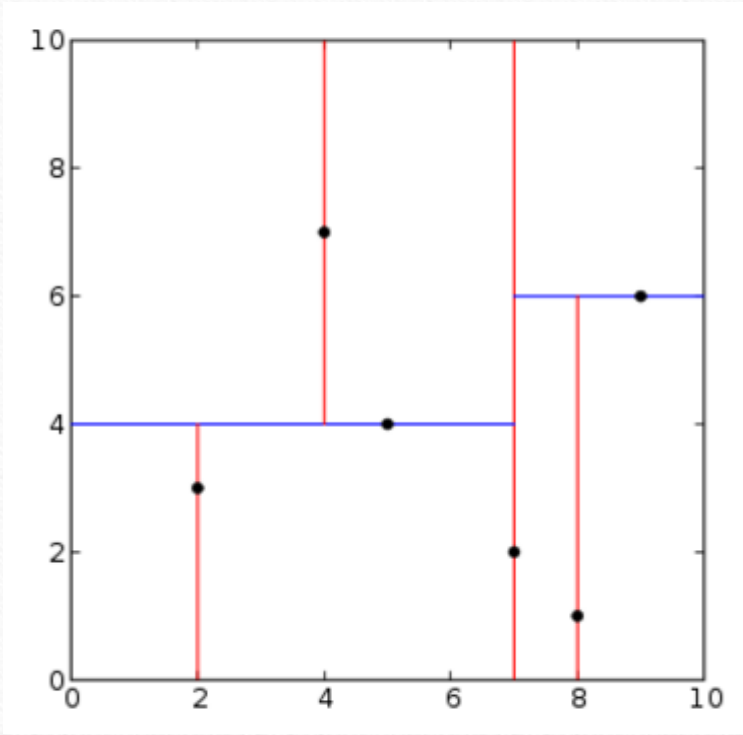
3. Efficient Implementation of k-means

3.1 K- Dimensional Tree:

Ex: k = 2;

K-d tree decomposition for the point set
(2,3) , (5,4) , (9,6) , (4,7) , (8,1) , (7,2)





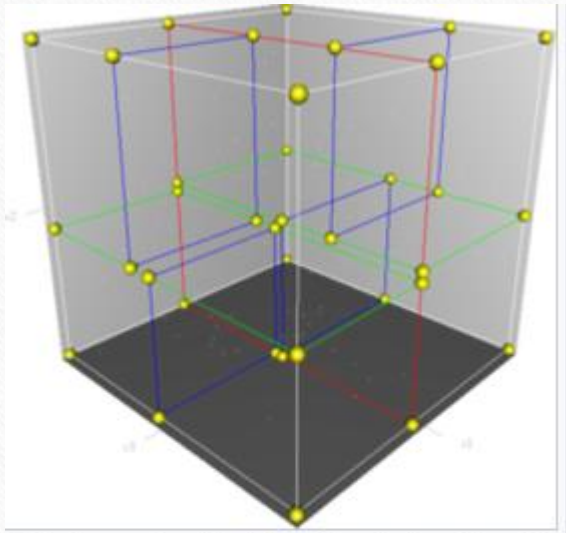
3. Efficient Implementation of K-means

3.2 Storing data points in kd-tree:

- Algorithm stores the multi-dimensional data points in a kd-tree
  - ❖ Box → axis-aligned hyper-rectangle
  - ❖ Bounding Box of a point set→ smallest box containing all the points

- Each Node of the kd-tree is associated with a closed box called a cell C
  - ❖ Root node's cell C is a bounding box of the given point set

- If the cell contains at most 1 point , then it is a **leaf**
- Otherwise split into two hyper rectangles by an axis-orthogonal hyperplane passing through the median of the associated data points



A 3-dimensional *k*-d tree. The first split (the red vertical plane) cuts the root cell (white) into two subcells, each of which is then split (by the green horizontal planes) into two subcells. Finally, those four cells are split (by the four blue vertical planes) into two subcells. Since there is no more splitting, the final eight are called leaf cells.

3.3 Pruning/ Filtering Algorithm:

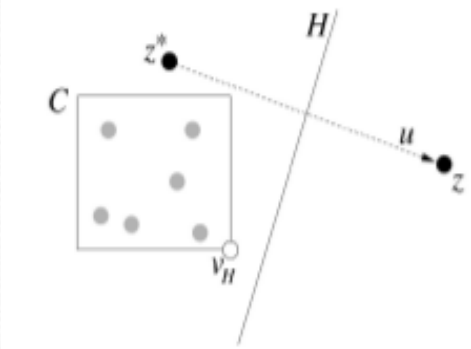
For each node of the kd-tree , we maintain a set of candidate centers Z.
Z → subset of center points that might serve as nearest neighbor for some point lying within the associated cell
z* → closest point in Z to C's midpoint

For each (z belongs to Z \ {z*})

- ❖ If no part of C is closer to z than it is to z* , then **z** can be **pruned** or **removed from the Z**

Determining if z is closer to any point in Z than from z* to that point:

- ❑ If cell lies entirely on one side of the bisecting plane H then the center z lying on the other side is pruned or removed from the set of candidate centers.
- ❑ If cell lies partially on the both sides of bisecting hyper plane H, then the both centers lying on either sides are retained. No pruning takes place.



For each internal node u in the tree,
u.Count ← associated data points
u.wgtCent ← vector sum of all the associated points
Actual Centroid ← u.wgtCent/u.count.

4. Results:

- This Filtering algorithm significantly outperformed the Brute Force and kd-center algorithms in all the cases for both synthetic data and real data.
- The experiments have shown that the filtering algorithm significantly outperforms the brute force and kd-center algorithm in all the cases provided the dimensions are moderate between 1 to 20
- As the cluster separation increases, the running time of the algorithm decreases.
- Filtering Algorithm's running time increases exponentially with dimension.

5. Pros:

- In this Algorithm, kd-tree data structure does not need to be recomputed at each stage
- It is simple and efficient and does better than Brute force and kd-center approach

6. Cons:

- Algorithm might encounter scenarios in which it degenerates to brute-force search
- Filtering algorithm's running time increases exponentially with dimension. Thus, as the dimension increases, the speed advantage would tend to diminish

7. Future Work:

- In the later stages of Llyod's algorithm as the centers are converging to their final positions, the most number of the data points have the same closest center from one stage to the next
- Filtering algorithm does not exploit this coherence
- A Kinetic method was proposed along these lines but this algorithm is quite complex and does not have faster running time in practice.
- Hence the Future work would be to combine the best elements of kinetic and filtering approaches.

8. Work planning to take up:

- I'm planning to implement this efficient k-means Lloyd's Filtering algorithm using kd-tree data structure to store the multi-dimensional data points. Hence Faster running time can be achieved as the data structure does not have to be recomputed at each stage and also because of
- Filtering or pruning the candidate set of the nodes as they are propagated down the kd-tree.
- I'm also planning to use kmeans++ to initialize the cluster centers before proceeding with the standard k-means optimization iterations.