Weekly Report 2 - K-Means

Ganji Varshitha
AI20BTECH11009

Introduction

K-Means is an unsupervised learning algorithm which performs partitioned clustering. Clustering helps us to understand the structure of the data by grouping it into distinct sub-groups.

Algorithm

It is a parametric method where we need to specify the number of clusters K, which we want to divide the data into.
It is a simple and iterative algorithm described as follows:

> Initialize K random points from the dataset as centroids
>
> **repeat**
>
> > Form K clusters by assigning all points to the closest centroid
> >
> > Recompute the centroid of each cluster
>
> **until** The centroids don't change

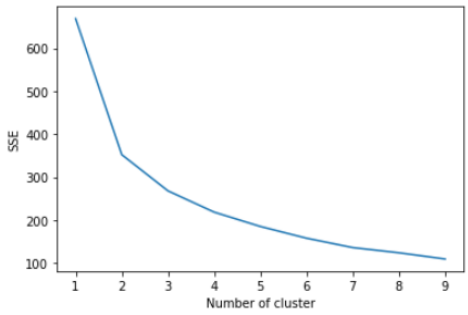
It can be seen as Expectation-Maximisation problem where the E-step is assigning the data points to the closest cluster and the M-step is computing the centroid of each cluster. It minimises the intra-cluster sum of squared distance from its centroid and keeps the clusters distant from each other.

Evaluation metric

SSE - Elbow method

Since it is an unsupervised learning model, we don't have ground truth values to evaluate its performance. Hence, we select the model with a K value which minimises sum of squared distances between data points and its centroids.
When SSE is plotted against K, select the K where the graph flatten out and forms an elbow. This method is referred to as elbow method.

Figure 1: Elbow method plot example



Code Example

Generated data using gaussian distribution and took user defined values for K and input dimensions.

Data generation

In [1]:
```
NUM_PTS=3
#Accepting input dimensions(p) and number of clusters(k)
Y=np.empty((p,k*NUM_PTS),int)
p=int(input("Enter input dimensions: "))
k=int(input("Enter number of clusters:"))
#contains the data points of all the clusters
#declaring positive semi definite covariance matrix
cov=[]
y=[]

#Generating clusters using Gaussian distribution
for i in range(k):
  #declaring mean matrix
  mean=np.random.randint(i+1,p+i+1,p,int)
  cov=np.diag([i for j in range(p)])
  y=np.random.multivariate_normal(mean,cov,NUM_PTS).T
  Y=np.concatenate((Y,y),axis=1)
```

Declaring stopping condition and implementing the algorithm

In [2]:
```
#Setting stopping condition
#epsilon = 0.001
# Initialize error to a large value
#error = 10000.0
# Initialize centroids - assume 4 of them from a 2D Gaussian
   distribution (zero mean, unit variance)
D = np.random.multivariate_normal(np.zeros(p), np.identity(p), k).T
```

```
# Initialize iteration count to 0
count = 0
# Initialize cluster size to 0
num=np.zeros(k)
# Initialize centroid update to 0
centr= np.zeros((p, k))

#Initialise distance_vector
distance_vector=[]


while (error > epsilon):

  # Update clusters based on distance
  for idx in range(k*NUM_PTS):
    cur_y = Y[:,idx]
    print(cur_y)
    for j in range(k):
      cur_d=D[:,j]
      d1=np.sqrt(np.sum((cur_y-cur_d)**2))
      distance_vector.append(d1)
    # Find closest centroid
    min_idx = distance_vector.index(np.min(distance_vector))
    for i in range(k):
      if (min_idx==i):
        num[i]+=1
        for j in range(p):
          centr[j,i]+=cur_y[j]
          if (num[i]!=0):
            centr[j,i]=centr[j,i]/num[i]

  error=np.sqrt(np.sum((centr-D)**2))
  for i in range(k):
    for j in range(p):
      D[j,i]=centr[j,i]
  count+=1
```

Key points

- The algorithm clusters the data into distinct sub groups which will not work for overlapping clusters.
- Normalization is required as it deals with distances.
- It assumes spherical shapes of clusters with center as centroid and fails in case of complex designs or even elliptical shape.
- The algorithm works well with large datasets and easy to implement.
- The centroids are sensitive to outliers.
- The algorithm is stochastic as the clusters formed depend on centroid initialisation.

Questions

1. Does the cost function of the algorithm has an optimized solution? **Solution:** Objective function is given by

$$\arg\min \sum_{i=1}^{K} \sum_{x \in S_i} ||x - C_i||^2$$

   This function is a NP hard problem. Therefore, we only use Lloyd algorithm to stop iterating if the problem converges.
   Since most of the convergence takes place in first few iterations, stopping condition is changed to 'Until relatively few points change clusters'.

2. Illustrate an example where K means fails when clusters have complex design. **Solution:**

Figure 2: K means for non spherical clusters