CSE 601 - Project 2: Clustering Algorithms

Team members:

Saketh Varma Pericherla - **sakethva**

Arvind Thirumurugan - **athirumu**

Vijay Jagannathan - **vijayjag**

K-means:

Algorithm Description:

Result Visualization:

Result Evaluation:

Hierarchical Agglomerative clustering with Min approach:

Algorithm Description:

Result Visualization:

Result Evaluation:

DBSCAN:

- Density Based Spacial Clustering of Applications with Noise or DBSCAN is a clustering algorithm that seperates data points based on the density of the region they belong to.
- Clusters are defined as dense regions in the data space, separated from regions of lower density.
- The size of the region is defined by the ε-Neighborhood which gives the region within a radius of ε from a data point.
- Before jumping into the algorithm, some of the key concepts are described below:
- **Core point** - A point is a core point if it has more than a specified number of points (MinPts) within Eps —These are points that are at the interior of a cluster.
- **Border point** - A border point has fewer than MinPts within Eps, but is in the neighborhood of a core point.
- **Noise point** - A noise point is any point that is not a core point nor a border point.
- **Directly density-reachable** - An object q is directly density-reachable from object p if p is a core object and q is in p's ε-neighborhood.

Algorithm:

For each point in the dataset that is not yet classified do the following steps:

- Check if the point is a core point or not.
- If the point is not a core point label it as a noise point
- If the point is indeed a core point, collect all the points that are density reachable from the current point and label them as a new cluster
- Repeat the above steps until all the points are visited and labeled into a cluser or as noise.

Implementation:

1. The parameters ε and MinPts are specified when running the program. We choose a small epsilon value typically between 0.5 to 1.5 and MinPts value as 3 for optimal results.

2. Using the 'regionQuery' function find the neighbors of the point using ε and MinPts and euclidean distance, for each unvisited point in the matrix.

3. If the number of neighbors is more than the minimum number of points, classify the point to the corresponding cluster id.

4. If the number of neighbors is less than the minimum number of points, classify the label of that point to be -1.

5. The 'expandCluster' function is used to assign every unvisited point in the neighborhood of point to the same cluster id. The point's neighbors are calculated by 'regionQuery' and then added on to the existing neighboring points.

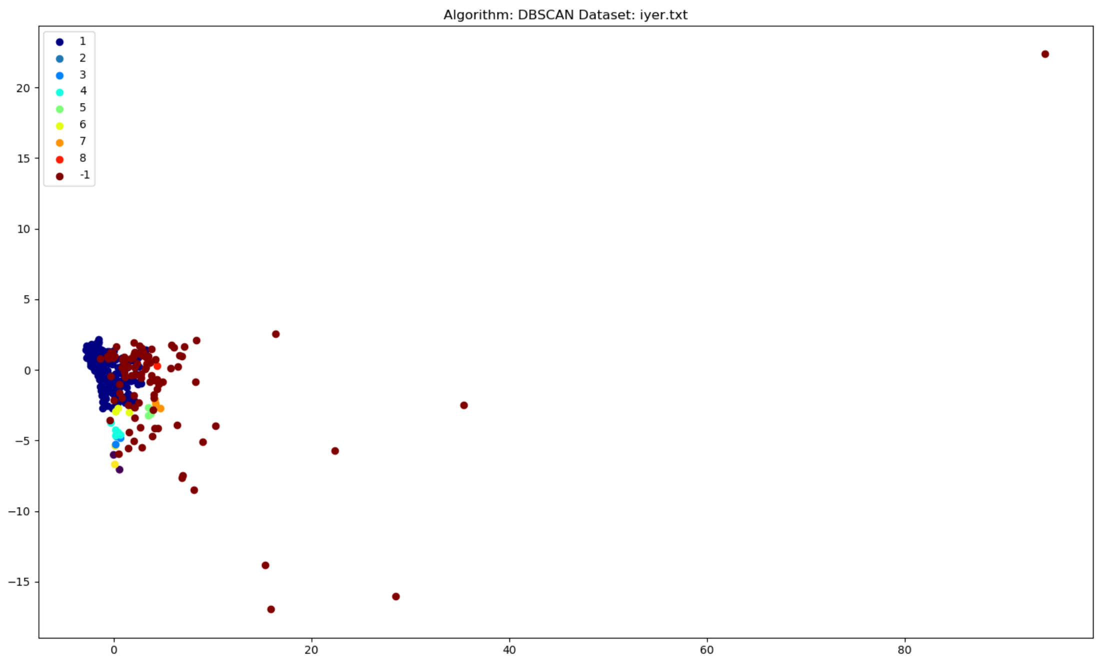
Result Visualization:

iyer.txt:

Epsilon: 1.3 MinPts: 3

Rand index: 0.6526755683922646

Jaccard index: 0.2835567491646023

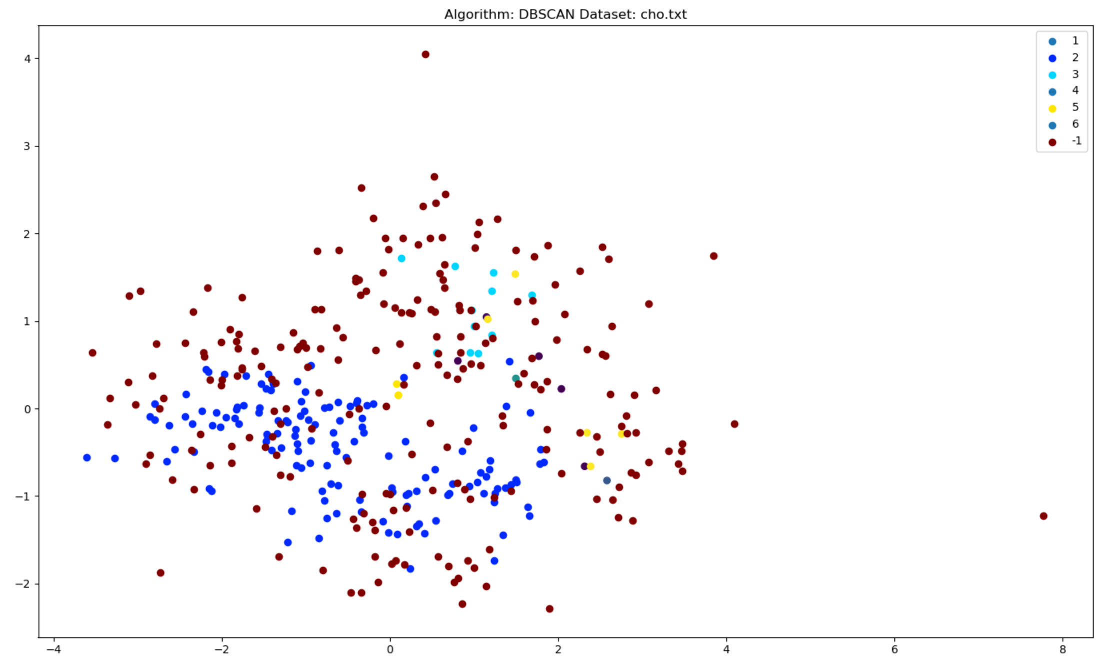


cho.txt:

Epsilon: 1 MinPts: 3

Rand index: 0.5664715831297484

Jaccard index: 0.2037425112793077



Result Evaluation:

- The time complexity of this algorithm is O(n^2), so it is reasonably fast when compared to algorithms like HAC and Spectral Clustering.
- If dataset is too large small clusters are likely to be labeled as noise
- If dataset is too small, even a small number of closely spaced that are noise or outliers will be incorrectly labeled as clusters.
- DBSCAN performs exceptionally well on datasets having well seperated dense regions of arbitrary shapes and sizes as demonstrated by the high jaccard value for the demo dataset while it performed poorly with iyer and cho datasets as a fixed epsilon value cannot account for the sparseness and disparity in distribution of the former two datasets.

Advantages:

- DBSCAN can find arbitrarily sized and shaped clusters.
- We don’t need to specify the number of clusters, as opposed to k-means.
- DBSCAN is resistant to outliers, as opposed to k-means.

Disadvantages:

- DBSCAN cannot perform well with data spread across varying densities since ε and MinPts are provided as constants for the entire dataset.
- DBSCAN is not completely deterministic. Depending on the order of processing the data, border points reachable from more than one cluster can be part of different clusters across multiple runs of the same algorithm.

Guassian Mixture Model:

- A mixture model is an probabilistic unsupervised model for clustering applications.
- In a Guassian Mixture model we use multiple Probability Density Functions to model the data into clusters.
- Unlike other clustering algorithms which do hard clustering where every point belongs to exactly one cluster, GMM performs soft clustering where every point belongs to several clusters with certain degrees.
- We use Expectation Maximization framework to train the model using GMM.

Implementation:

1. While running the program we specify number of clusters which is a required parameter and optional parameters like the initial mean, variance, convergenece threshold, maximum iterations and smoothing value as input to the program.
2. **Expectation:** For the given parameter values we predict the values of the latent variables. We use sklearn's multivariate_normal.logpdf function to calculate the probability distribution function to be multiplied with the probability parameter pi to obtain a prediction for each cluster. Below code snippet shows the impementation:

```
    for j in range(N):
        for k in range(self.num_clusters):
            pdf = multivariate_normal.pdf(
                self.X[j], mean=mu[:, k], cov=sigma[k])
            self.predicted[j, k] = pi[k] * pdf
        self.predicted[j] /= np.sum(self.predicted[j])
```

3. **Maximization:** The objective is to maximize the log likelihood function. We use the smoothing value to avoid singular matrix errors and use the multivariate_normal.logpdf function available in sklearn package to determine the log likelihood value. We then update the mean and variance with the newly obtained values. Below snippet shows the steps involved in maximization:

```
def log_likelihood(self, mu, sigma, pi):
   ...
    for n in range(N):
        for k in range(self.num_clusters):
            np.fill_diagonal(
                sigma[k], sigma[k].diagonal()+self.smoothing_value)
            loss += self.predicted[n, k]*math.log(pi[k])
            loss += self.predicted[n, k] * \
                multivariate_normal.logpdf(
                    self.X[n], mean=mu[:, k], cov=sigma[k])
    return loss
...
curr_likelihood = self.log_likelihood(mu, sigma, pi)

pred_sum = np.sum(self.predicted, axis=0)
pi = pred_sum / N
mu = np.dot(self.X.T, self.predicted) / pred_sum
sigma = np.zeros((self.num_clusters, M, M))
...
```

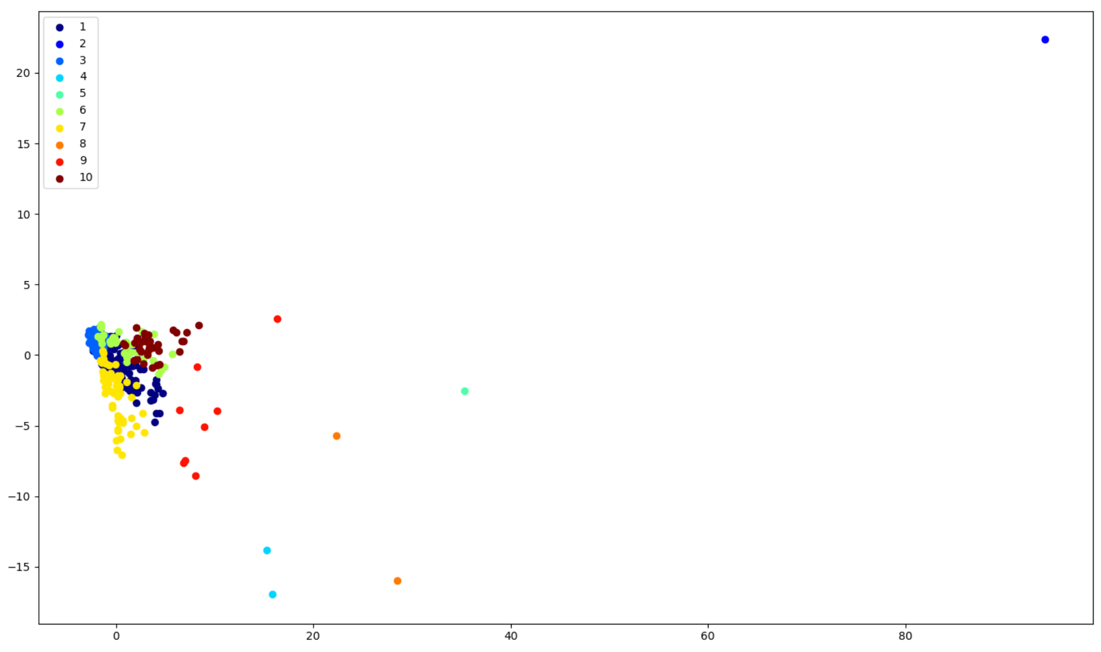
Result Visualization:

iyer.txt

Mean, covariance: Initialized using labels from K-means Number of clusters: 10 Smoothing threshold: 5*10^{-8}

Rand index: 0.7618083796938894

Jaccard index: 0.3529943801384132

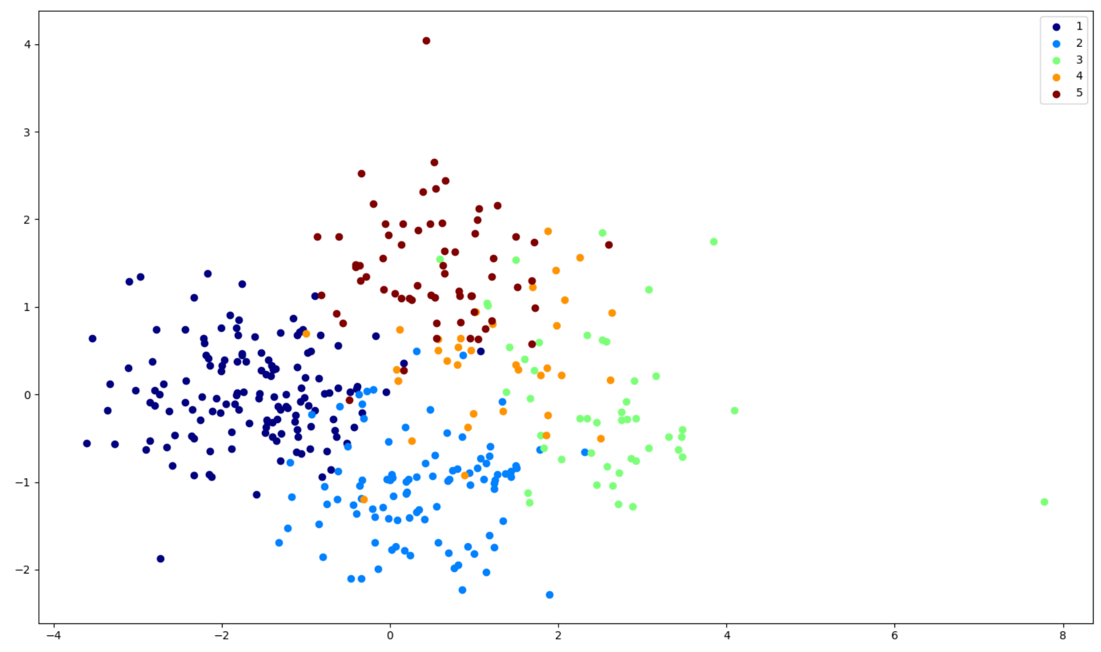


cho.txt

Mean, covariance: Initialized using labels from K-means Number of clusters: 5 Smoothing threshold: 10^{-15}

Rand index: 0.7959005610888883

Jaccard index: 0.40182540619221844



Result Evaluation:

- GMM has provided more accurate results than density based given that it is initialized by K-means and the fact that it uses probabilistic assignments instead of hard assignments based on near neighbor density.
- Though density based clustering is not accurate for sparsely seperated cho dataset and disparity in distribution of points in iyer dataset, GMM is able to provide more accuracy leveraging multiple Guassians and probabilistic assignments.

Advantages:

- GMM can handle clusters with varying sizes and variance
- GMM gives probabilistic cluster assignments so that soft clustering can be performed for more accurate decision making. In contrast, density based clustering can wrongly assign border points.

Disadvantages:

- Initialization is crucial for generating accurate clusters using GMM. Other clustering algorithms like K-means are used to initialize the mean and variance before running GMM.
- GMM has overfitting problems where the algorithm tries to fit the data points perfectly and ends up performing poorly on new data.

Spectral Clustering:

Algorithm Description:

Result Visualization:

Result Evaluation: