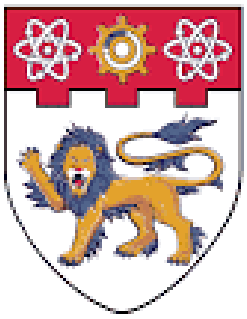


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

Project 1 Technical Review on Clustering Methods

Group 24

Name	Matric No.
Singh Ananya	U1923401D
Goyal Bhavya	U1923690L
Kenny Tan Junrong	U1920056G
Gupta Suhana	U1923230B
Gupta Ameeshi	U2023994H

TABLE OF CONTENT

1. Abstract

The emergence of big data has made clustering a key component in data analytics & mining. Clustering refers to grouping data points together because of their similar characteristics. In other words, divide the points with similar traits and assign them into clusters. It helps us to understand more about the distribution of data, which allows us to observe and analyze the characteristics of each cluster. In this review paper, we will be evaluating the various clustering methods on 2 main datasets (MNIST and IRIS) based on their strengths and weaknesses. At the same time, by researching and discussing in-depth, we can determine which method works best in different scenarios.

2. Introduction

In this review paper, we will be covering the following unsupervised clustering methods:

1) K-Means Clustering
2) Gaussian Mixture Model (GMM)
3) DBSCAN Clustering
4) Agglomerative Clustering
5) Deep Embedded Clustering

The datasets that we will be using are MNIST and IRIS datasets. The MNIST dataset holds 70000 examples of handwritten digits, with a training set of 60,000 examples and a test set of 10,000 examples. On the other hand, the IRIS dataset consists of 150 examples of iris plants with 3 classes of 50 instances from each species. Both of the datasets are useful and beneficial to those who want to work on learning techniques and pattern recognition methods on real-world data as less effort is required for preprocessing and formatting of the data.

For this review paper, we will be using IRIS dataset to compare GMM and DBSCAN Clustering against K-Means clustering to learn about their strengths and weaknesses. We will be assessing their performance based on the following metrics (scikit-learn):

- **Accuracy Score** - Number of data points that have been assigned to the correct clusters against the total number of data points in the dataset.
- **Silhouette Score** - To evaluate the goodness of the clustering method,

$$Silhouette\ Score\ = \frac{(b-a)}{max(a,b)}$$

where a = average distance between each point within a cluster,

b = average distance between all clusters

The score varies from -1 to 1. A score of 1 means that the clusters are well separated and distinct, 0 means that the clusters are overlapping and -1 means that clusters are assigned wrongly.

- **Adjusted Random Score** - To measure similarity between two clusterings by performing pair-wise comparisons with pairs that are assigned in the same or different clusters in the predicted and true clusterings.

$$ARI\ = \frac{RI\ - E[RI]}{max(RI) - E[RI]}$$

where RI = random index, E[RI] = expected random index

The score varies from -1 (random labellings) to 1 (perfect match). The higher the score, the better.

- **Adjusted Mutual Info Score** - Given the knowledge of *U*, class assignments of the actual data and *V*, class assignments of the clustering algorithm, it measures the agreement of the 2 assignments.

$$AMI = \frac{MI(U,V) - E(MI(U,V))}{avg(H(U), H(V)) - E(MI(U,V))}$$

- **Davies Bouldin Score** - To measure the similarity of each cluster with a cluster most similar to it.

  S_i is the average distance between feature vectors in clusters i and centroid of the cluster

$$S_i = \left(\frac{1}{T}\sum_{j=1}^{T_i} ||X_j - A_i||_p^q\right)^{1/q}$$

  $M_{i,j}$ is the measure of separation between cluster C_i and cluster C_j

$$M_{i,j} = ||A_i - A_j||_p$$

  $R_{i,j}$ is the measure of how good the clustering scheme is

$$R_{i,j} = \frac{S_i + S_j}{M_{i,j}}$$

$$D_i \equiv max_{j \neq i} R_{i,j}$$

$$DB\ Index \equiv \frac{1}{N}\sum_{i=1}^{N} D_i$$

  where N = number of clusters

  Best score will be 0 which represents that the clusters are distinct. The lower the score, the more distinct the clusters.

- **Calinski Harabasz (CH) Score** - also known as Variance Ratio Criterion which is defined as the ratio of the sum of between-cluster dispersion and of within-cluster dispersion.

$$CH\ Index\ = \frac{BGSS}{WGSS} \times \frac{N-K}{K-1}$$

where BGSS = between-group sum of squares,

WGSS = within-group sum of squares,

N = total number of data points,

K = total number of clusters

Larger values of CH index represent better clustering.

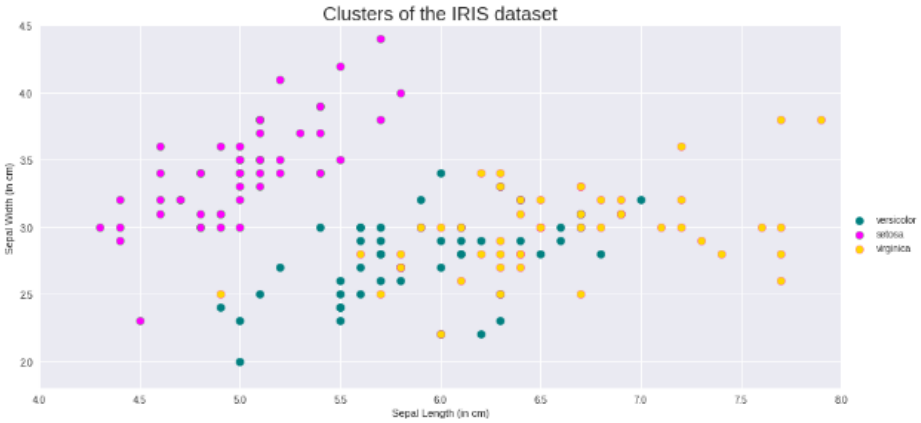


Fig 1: Scatter plot for the Iris Dataset (Ground Truth)

MNIST dataset allows us to widen our research and look into image clustering. We will be using the MNIST dataset to assess the effectiveness of Agglomerative Clustering and Deep Embedded Clustering against K-Means Clustering. We will be evaluating them based on:

- **Accuracy** :

$$ACC = \max_{m} \frac{\sum_{i=1}^{n} \mathbf{1}\{l_i = m(c_i)\}}{n}, \qquad (10)$$

where l_i is the ground-truth label, c_i is the cluster assignment produced by the algorithm, and m ranges over all possible one-to-one mappings between clusters and labels.

- **Number of clusters** : The no. of clusters each algorithm divides the data points into for the MNIST dataset.

3. Related Works

Clustering has been extensively studied in machine learning in terms of feature selection (Boutsidis et al., 2009; Liu & Yu, 2005; Alelyani et al., 2013), distance functions (Xing et al., 2002; Xiang et al., 2008), grouping methods (Mac- Queen et al., 1967; Von Luxburg, 2007; Li et al., 2004), and cluster validation (Halkidi et al., 2001). Space does not allow for a comprehensive literature study and we refer readers to (Aggarwal & Reddy, 2013) for a survey.

One branch of popular methods for clustering is K-Means (MacQueen et al., 1967) and Gaussian Mixture Models (GMM) (Bishop, 2006). These methods are fast and applicable to a wide range of problems. However, their distance metrics are limited to the original data space and they tend to be ineffective when input dimensionality is high (Steinbach et al., 2004).

Several variants of K-Means have been proposed to address issues with higher-dimensional input spaces. De la Torre & Kanade (2006); Ye et al. (2008) perform joint dimensionality reduction and clustering by first clustering the data with k-means and then projecting the data into lower dimensions where the inter-cluster variance is maximized. This process is repeated in EM-style iterations until convergence. However, this framework is limited to linear embedding; our method

employs deep neural networks to perform non-linear embedding that is necessary for more complex data.

Spectral clustering and its variants have gained popularity recently (Von Luxburg, 2007). They allow more flexible distance metrics and generally perform better than k-means. Combining spectral clustering and embedding has been explored in Yang et al. (2010); Nie et al. (2011). Tian et al. (2014) proposes an algorithm based on spectral clustering, but replaces eigenvalue decomposition with deep autoencoder, which improves performance but further increases memory consumption. Most spectral clustering algorithms need to compute the full graph Laplacian matrix and therefore have quadratic or super quadratic complexities in the number of data points. This means they need specialized machines with large memory for any dataset larger than a few tens of thousands of points. In order to scale spectral clustering to large datasets, approximate algorithms were invented to trade off performance for speed (Yan et al., 2009). Our method, however, is linear in the number of data points and scales gracefully to large datasets.

Minimizing the Kullback-Leibler (KL) divergence between a data distribution and an embedded distribution has been used for data visualization and dimensionality reduction (van der Maaten & Hinton, 2008). T-SNE, for instance, is a non-parametric algorithm in this school and a parametric variant of t-SNE (van der Maaten, 2009) uses deep neural network to parametrize the embedding. The complexity of t-SNE is O(n2), where n is the number of data points, but it can be approximated in O(n log n) (van Der Maaten, 2014).

We take inspiration from parametric t-SNE. Instead of minimizing KL divergence to produce an embedding that is faithful to distances in the original data space, we define a centroid-based probability distribution and minimize its KL divergence to an auxiliary target distribution to simultaneously improve clustering assignment and feature representation. A centroid-based method also has the benefit of reducing complexity to O(nk), where k is the number of centroids.

4. Methods

4.1 K Means

K-means is a centroid-based algorithm or a distance-based algorithm where the distance of the data points to cluster centroids are calculated and the point that is within the minimum distance of a specific cluster will be assigned to the cluster. Each cluster is associated with a centroid. The goal of K-Means algorithm is to minimize the Within Cluster Sum of Square Distance (WCSSD) between the points and their cluster centroids. This is the pseudocode for K-means algorithm:

1: Select the number of clusters k to assign
2: Choose k data points randomly as centroids of the clusters
3: Repeat until the centroid positions do not change
   a. *Assign the data points to clusters that are closest to their respective centroids*
   b. *Compute the new centroid of the newly formed clusters*

4.1.1 Experiments with K-Means using IRIS dataset

Choosing the number of clusters

To choose the optimal number of clusters for the dataset, we plotted the Within Cluster Sum of Squared Distances (WCSSD) of the data points to their closest cluster center. In order to choose the best number of clusters we see if the plot looks like an arm and then choose the elbow on the arm as the optimal number of clusters. We observe that with increase of k, the WCSSD decreases gradually but we would like to select the point where the drop in the graph is substantial at a low cluster number, so that the model can be generalized to unseen data and also avoid the overhead of having many clusters.

From Figure 2 below, we can see that the elbow point for the iris dataset is at 3 clusters, which seems valid as we know that the iris dataset has three classes.

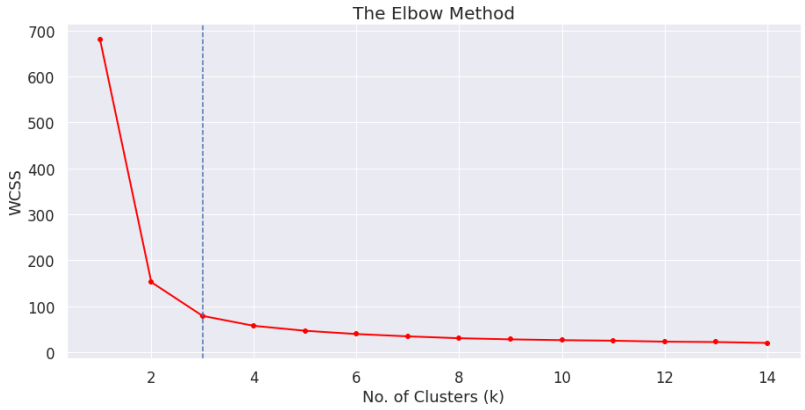


Fig 2: Elbow Curve for the Iris Dataset

Below is the result after classifying the dataset into 3 clusters:

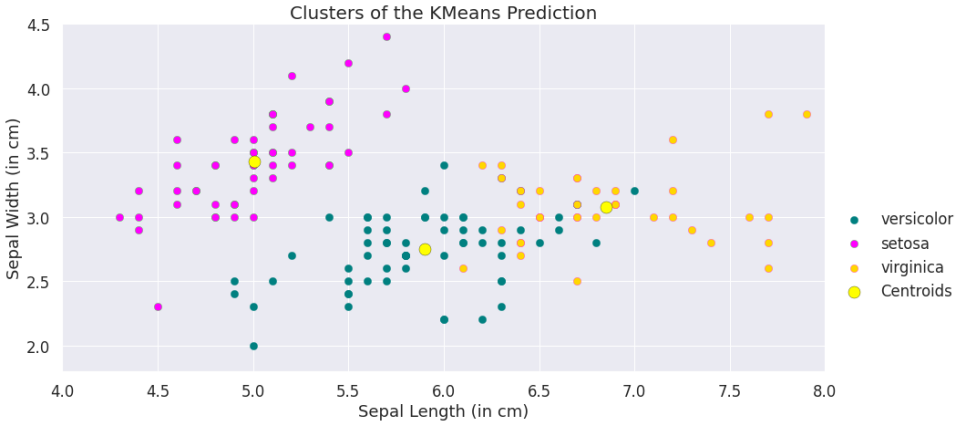


Fig 3: Clustering results for Iris Dataset

Metric	**Score**
Accuracy Score	0.893

Silhouette Score	0.735
Adjusted Random Score	0.730
Adjusted Mutual Info Score	0.755
Davies Bouldin Score	0.661
Calinsku Harabasz Score	561.627

Table 1: K-means clustering score on Iris Dataset

The number of iterations of k-means algorithm

The sklearn library gives us an option to specify the n_init parameter for a run of k-means. This parameter signifies the number of times the k-means algorithm will be run with different centroid initializations. The final result will then be the best output of the n_init consecutive runs in terms of the WCSSD calculated by the algorithm internally.

Metric	**Score (n_init = 10)**	**Score (n_init = 1)**
Accuracy Score	0.893	0.92
Silhouette Score	0.735	0.696
Adjusted Random Score	0.730	0.785
Adjusted Mutual Info Score	0.755	0.775
Davies Bouldin Score	0.661	0.692
Calinsku Harabasz Score	561.627	539.188

Table 2: Comparison of different iterations on Iris Dataset

Different algorithms

Elkan Algorithm: This algorithm accelerates the discovery of clusters in the K-Means algorithm while giving exactly the same result as the standard algorithm. It uses triangle inequality to avoid unnecessary distance calculations by keeping track of the lower and upper bounds for distances between points and centers. (Elkan 2003) This algorithm is much more effective with a higher number of clusters and with higher dimensional data. Since the Elkan algorithm gives the same result as the default run (with "full" as the algorithm parameter) of K-Means, the only difference we observed was for the time. The Elkan algorithm took less time to train (approx 0.0007s) the k-means on the Iris Dataset with 3 clusters, as compared to the default training (approx 0.001s) for k-means. Below are the results obtained while using the Elkan algorithm for training k-means.

Metric	**Score (using Elkan)**
Accuracy Score	0.893
Silhouette Score	0.735
Adjusted Random Score	0.730
Adjusted Mutual Info Score	0.755
Davies Bouldin Score	0.661
Calinsku Harabasz Score	561.627

Table 3: K-means clustering score using Elkan Algorithm on Iris Dataset

K-Means++: This algorithm is a variation of K-Means where we initialize the centroids in a smarter way so that it can give better results compared to random centroid initializations. In this method, we pick the first centroid point randomly and then select a point which is farthest from the picked up point. We make this new point as the new centroid and then repeat this step until we find k centroids. This way all the points selected have maximum probability proportional to distances between them. Below are the results for the cases when we use K-Means++ and random initializations of centroids.

Metric	**Score (K-Means++)**	**Score (Random)**

Accuracy Score	0.893	0.746
Silhouette Score	0.735	0.681
Adjusted Random Score	0.730	0.537
Adjusted Mutual Info Score	0.755	0.635
Davies Bouldin Score	0.661	0.621
Calinsku Harabasz Score	561.627	418.175

Table 4: Comparison of different algorithms on Iris Dataset

4.1.2 Experiments with K-Means using MNIST dataset

Steps taken to apply K-Means on dataset

Preprocessing:

Reshaping: The original MNIST dataset contains a 28x28x1 pixel image. Images stored as NumPy arrays are 2-dimensional arrays. However, the K-means clustering algorithm takes 1-dimensional arrays as input; as a result, we will need to reshape each image or 'flatten' the data. Clustering algorithms almost always use 1-dimensional data. MNIST contains images that are 28 by 28 pixels; as a result, they will have a length of 784 once we reshape them into a 1-dimensional array.

Normalization: In order to facilitate training, we need to convert each pixel value into 0 to 1 range. The maximum value of a pixel in grayscale is 255, so we can normalize it by dividing 255.

Applying K-Means Clustering:

Mini Batch K-Means: The primary idea behind the Mini Batch K-means method (Medewar, 2022) is to create small random batches of data with a predetermined size so that they may be

kept in memory. Each iteration obtains a fresh random sample from the dataset and uses it to update the clusters, and the process is continued until convergence. Each mini-batch updates the clusters using a convex mixture of the prototype values and the data, with a decreasing learning rate as the number of iterations increases. The inverse of the number of data assigned to a cluster during the procedure is this learning rate. Because the influence of incoming data diminishes as the number of iterations grows, convergence can be observed when no changes in the clusters occur for multiple iterations in a row.

Since the size of the MNIST dataset is quite large, we will use the mini-batch implementation of k-means clustering. This will reduce the amount of time it takes to fit the algorithm to the data. We choose the n_clusters argument to the n_digits (the size of unique labels, in our case, 10), and set the default parameters in MiniBatchKMeans.

Evaluating K-Means Clustering:

Since we are using this clustering algorithm for classification, accuracy is ultimately the most important metric; however, there are other metrics out there that can be applied directly to the clusters themselves, regardless of the associated labels. Two of these metrics that we use are inertia and homogeneity.

Inertia: Inertia measures how well a dataset was clustered by K-Means. It is calculated by measuring the distance between each data point and its centroid, squaring this distance, and summing these squares across one cluster. A good model is one with low inertia and a low number of clusters (K). However, this is a tradeoff because as K increases, inertia decreases. (Chouinard 2022 May 5)

Homogeneity: An entirely homogeneous clustering is one where each cluster has information that directs a place toward a similar class label. Homogeneity portrays the closeness of the clustering algorithm to this (homogeneity_score) perfection. Homogeneity Score is a value between 0.0 and 1.0, where 1.0 stands for perfectly homogeneous labeling. (Pooh ® 2018 Nov 3)

Earlier we made the assumption that K = 10 was the appropriate number of clusters; however, this might not be the case. Therefore, we fit the K-means clustering algorithm with several different values of K, and then evaluate the performance as shown in the table and figures below.

Number of Clusters	**Metric**		
	Accuracy	**Homogeneity**	**Inertia**
10	0.564	0.465	2383375.0
16	0.650	0.553	2208197.5
36	0.767	0.678	1961340.875
64	0.789	0.727	1822361.625
144	0.867	0.804	1635514.25
256	0.900	0.842	1519708.25

Table 5: Comparison of different number of clusters on MNIST Dataset

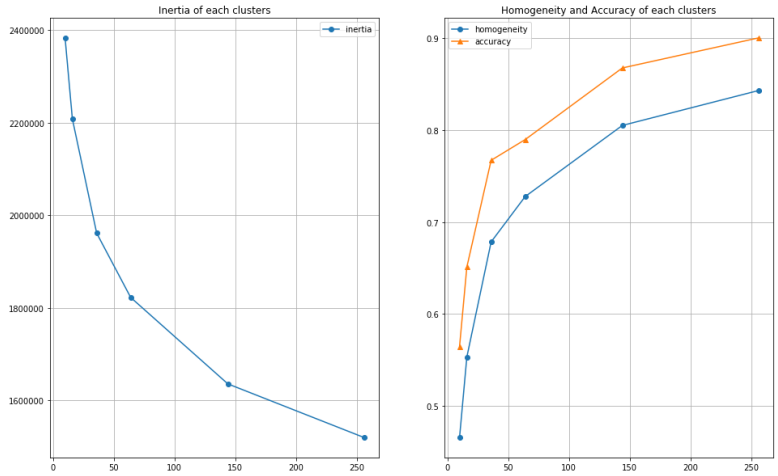


Fig 4: Graphs depicting Inertia, Homogeneity, and Accuracy as compared to Number of Clusters

As a result, we found out that when the K value is increased, the accuracy and homogeneity is also increased, however, the inertia decreases. We also checked the performance on the test

dataset. The MiniBatchKmeans Clustering model displayed almost 90% accuracy. One definite way to check the model performance is to visualize the real image. For convenience, we decreased the n_clusters to 36. The resulting image is shown below.

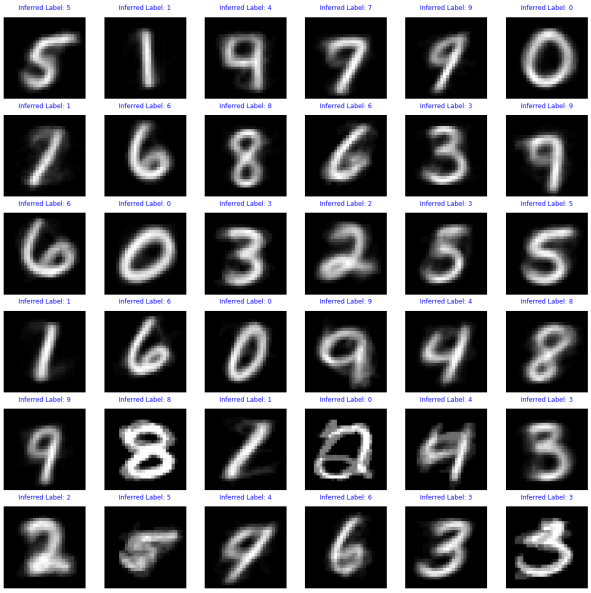


Fig 5: Real Output to analyze model performance

4.1.3 Strengths and Weaknesses of K-Means

Strengths	Weaknesses
- Guarantees convergence	- Choose k manually
- Easily interpretable	- Dependent on centroid initialization
- Relatively simple to implement	- Difficult to classify data of varying sizes and density
- Generalizes to clusters of different shapes and sizes	- Affected by outliers
	- Difficult to cluster data with higher dimensions

4.2 Gaussian Mixture Model (GMM)

GMM is a distribution-based algorithm, in other words, a probabilistic model where it assumes that all the data points are generated from a certain number of Gaussian distributions and each of the distributions represent a cluster. As a result, data points that belonged to a single distribution will tend to be grouped together. For a dataset with *m* features, we will have *k* Gaussian distributions (*k* represents the number of clusters), and each of them have their respective mean and variance value. To determine the mean and variance value for each Gaussian distribution, we will have to implement Expectation-Maximization (EM) which helps us find the right model parameters especially when the data has missing values or is incomplete. Thus, it is important in the implementation of GMM since GMM considers the cluster that the data point will be assigned to as unknown. The EM algorithm is carried out in 2 steps:

1. **Expectation Step**

   Calculate the probability of each data point *xi* belonging to the k clusters

$$\gamma_i = \frac{probability\ x_i\ belongs\ to\ c}{sum\ of\ probability\ x_i\ belongs\ to\ c_1,\ c_2,\ ..\ c_k}$$

$$= \frac{\pi_c\ \mathcal{N}(x_i\ ;\ \mu_c, \Sigma_c)}{\Sigma_{c'}\ \pi_{c'}\ \mathcal{N}\left(x_i ; \mu_{c'}, \Sigma_{c'}\right)}$$

Fig 6: Mathematical formula for Expectation Step

   If the data point is assigned to the right cluster, the value generated will be high.

2. **Maximization Step**

Update the Π (density), μ (mean) and Σ (covariance) values

$$\Pi = \frac{\text{Number of data points assigned to cluster}}{\text{Total number of data points}}$$

$$\mu = \frac{1}{\text{Number of data points assigned to cluster}} \Sigma_i \gamma_{ic} x_i$$

$$\Sigma_c = \frac{1}{\text{Number of data points assigned to cluster}} \Sigma_i \gamma_{ic} (x_i - \mu_c)^T \left(x_i - \mu_c\right)$$

Fig 7: Mathematical formula for Maximisation Step

Π is the ratio of the number of points in the cluster and the total number of data points.

μ and Σ are updated based on the values assigned to the distribution. To maximise the log-likelihood function, the updated values will be used to calculate the new probabilities for each data point and update the values repeatedly.

This EM algorithm is taken care of by the *GaussianMixture* object from *sklearn.mixture*.

4.2.1 Experiments with GMM using IRIS dataset

Choosing the optimal number of clusters

To assess the optimal number of clusters in the data, we implemented Akaike Information Criteria (AIC) and Bayesian information criteria. AIC can be termed as a measure of the goodness of fit of any estimated statistical model while BIC is an estimate function of the posterior probability that the model is true, with different numbers of parameters. In order to choose the best number of clusters, we plot the AIC and BIC score with different bars representing the different covariance types (spherical, tied, diagonal, full) against the number of components. We then selected the elbow of the figure, the point after which the decrease becomes notably smaller.

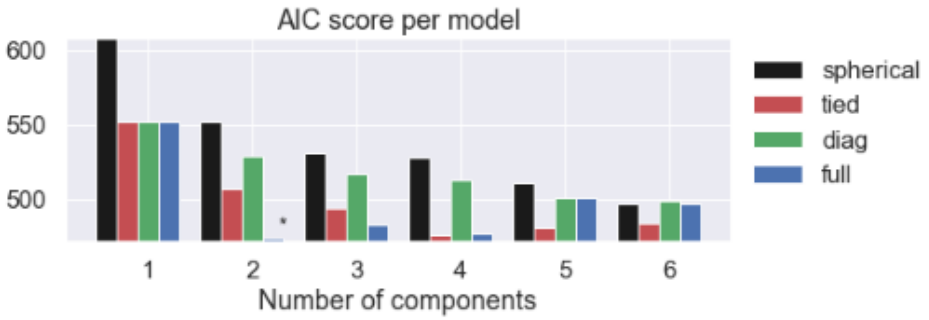


Fig 8: AIC Score grouped bar chart

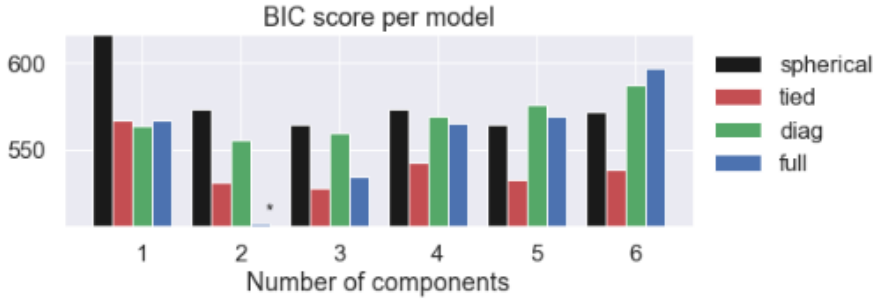


Fig 9: BIC Score grouped bar chart

From the AIC and BIC plot above, we can see that the ideal number of components that we should implement for GMM is 3. This is reasonable as the iris dataset has 3 different types of species. After which, GMM is implemented, data points are assigned to their respective clusters and the results are plotted.

Results of GMM

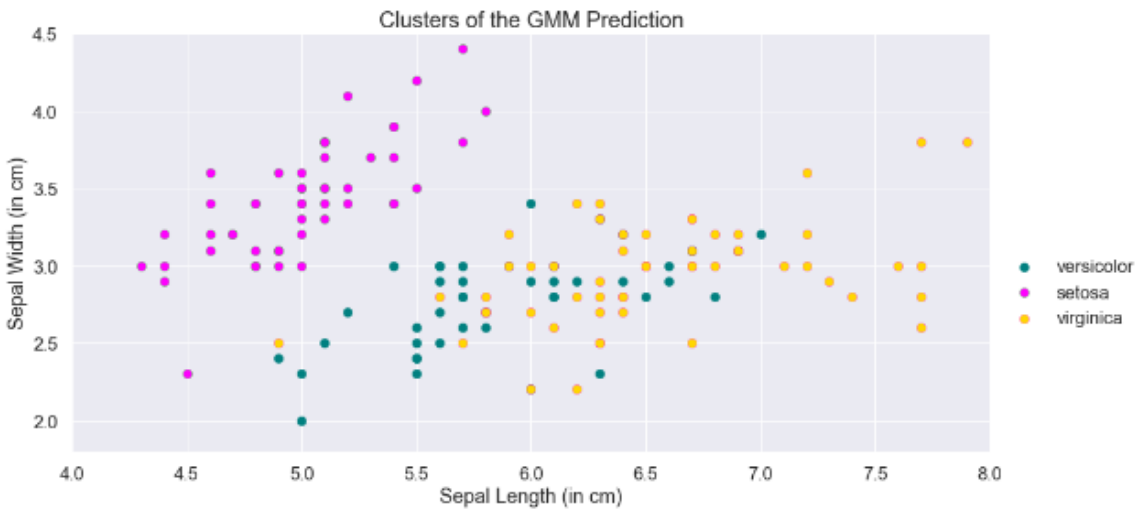


Fig 10: GMM Clustering results for Iris Dataset

From Figure 10 above, we can see that the prediction of clusters by GMM model is very similar to the actual plot of the clusters (species) of the data points. This is further supported by the high accuracy score that was calculated based on the results from GMM.

Metric	Score
Accuracy Score	0.967
Silhouette Score	0.650
Adjusted Random Score	0.904
Adjusted Mutual Info Score	0.898
Davies Bouldin Score	0.748
Calinsku Harabasz Score	481.781

Table 6: GMM Clustering metrics score for Iris Dataset

From the results, we can see how GMM is a better clustering model than K-Means and understand better why is it so. Firstly, the shape of the decision boundaries. K-Means makes fixed partitions which means it imposes a hard break between clusters. GMMs are more flexible with the soft clustering approach where they can be clustered with different covariant shapes that is extendable to fitting and returning overlapping clusters. Secondly, k-means only consider the mean to update the centroid while GMM takes into account the mean as well as the variance of

the data. Thirdly, K-Means groups data points based on their distance from the cluster centroids while GMM is a probabilistic algorithm where probabilities of the data points to their clusters are assigned. With the probability, we can be more confident that a given data point belongs to a specific cluster. Lastly, even though GMM takes a longer time than K-Means to run due to the iterations of the EM algorithm, GMM is more robust and performs better.

4.2.2 Strengths and Weaknesses of GMM

Strengths	Weaknesses
- Good at handling clusters with varying sizes, variance, etc. - Can handle missing data, effective for data with a lot of noise or not well-defined - Accommodates mixed membership based on their probabilities	- More difficult to initialize or train - Overfitting issues

4.3 Agglomerative Clustering

Agglomerative Clustering is a type of hierarchical clustering algorithm. It is an unsupervised machine learning technique that divides the population into several clusters such that data points in the same cluster are more similar and data points in different clusters are dissimilar. Agglomerative clustering uses a bottom-up approach, wherein each data point starts in its own cluster. These clusters are then joined greedily, by taking the two most similar clusters together and merging them. Clustimage was used for clustering, extracting the features, and evaluating the optimal number of clusters in the high-dimensional feature space. Silhouette score was used for the evaluation of the clustering algorithm. Here are the steps for agglomerative clustering:

1. Agglomerative clustering begins with N groups, each containing initially one entity
2. Then the two most similar groups merge at each stage until there is a single group containing all the data.

3. A binary tree called a dendrogram will represent the merging process. The initial groups (objects) are on the leaves (at the bottom of the figure), and we join them in the tree each time when two groups are merged.

For either type of hierarchical clustering, the data set X is partitioned into Q sets $\{H1, \ldots, HQ\}$. That is, if subsets Ci and Cj satisfy that $Ci \in Hm$, $Cj \in Hl$, and $m > l$, then either $C_i \subset C_j$ or $C_i \cap C_j = \emptyset$ for all $i \neq j$, $m, l = 1, \ldots, Q$ [3]. In other words, for two subsets of any hierarchical partitions, either one subset contains the other entirely or they are completely disjoint. *Dendrograms*, as in Fig. 13, provide an easy-to-understand way to represent the nested structure of the clustering. Hierarchical clustering works especially well with smaller data sets. Agglomerative algorithms, because they not only have to determine the best way to pair the clusters at each iteration but also when the clustering is complete, become more computationally expensive as more data points are considered. In some ways, divisive algorithms are more efficient. At each time step, the algorithm only needs to split each cluster into two in a way that satisfies some criteria, for example, a minimization of the sum of squares error. The clustering is complete when all clusters are split into singleton sets.

4.3.1 Experiments with Agglomerative Clustering

- On MNIST Dataset
- Clusteval for evaluation
- Hierarchical clustering dendrogram

4.3.2 Results

1. The dimensionality is reduced to 29 features

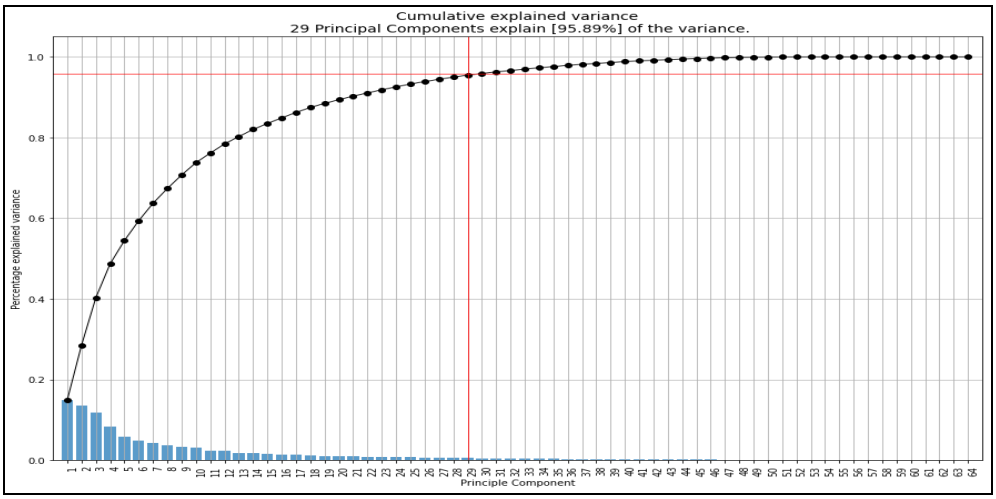


Figure 11: Cumulative explained variance plot

2. An optimum of 10 clusters is detected for the highest silhouette score

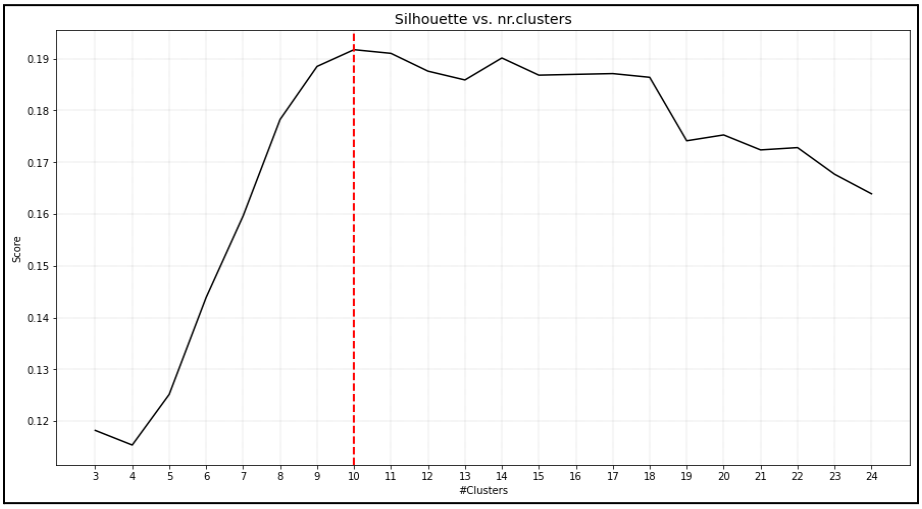


Fig 12: Silhouette vs Number of clusters

3. Dendrogram for the distribution of samples across the 10 clusters with an average silhouette score of 0.576

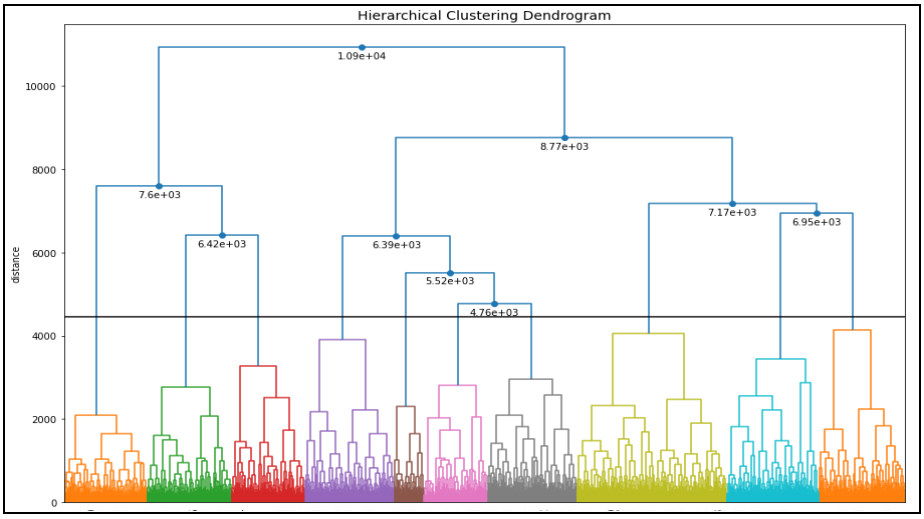


Fig 13: Hierarchical clustering dendrogram

4. Scatter plot of all clusters

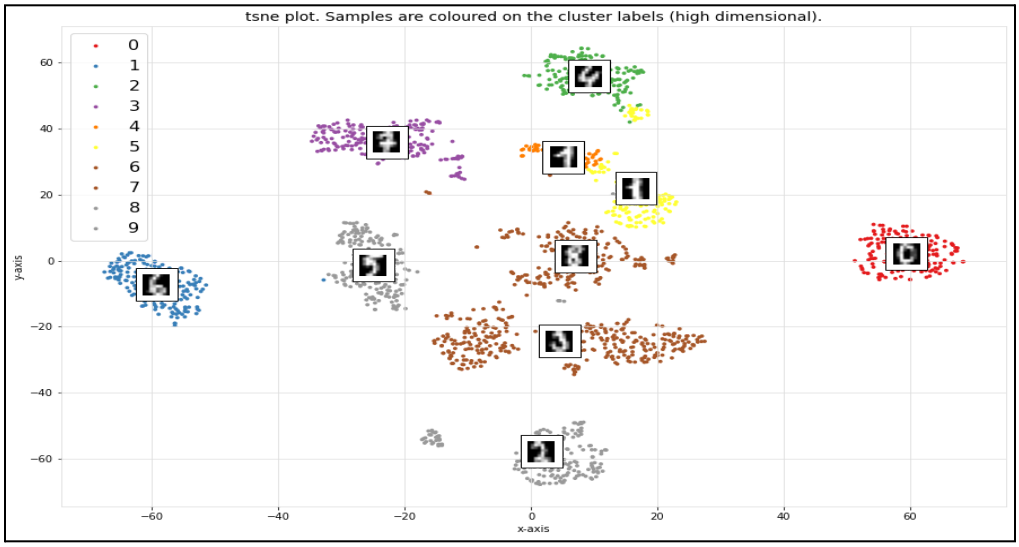


Fig 14: Scatter plot of all clusters

5. Unique image per cluster

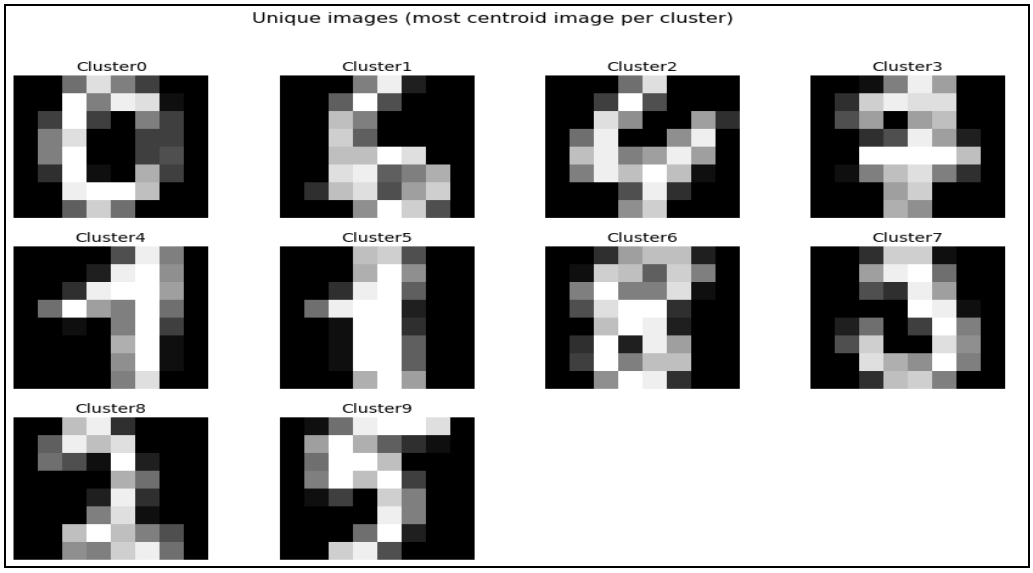


Fig 15: Unique image per cluster

4.3.3 Strengths and Weakness

Strength	Weakness
It works from the dissimilarities between the objects to be grouped together.	Groups with close pairs can merge sooner than is optimal

4.4 Deep Embedded Clustering

Relatively little work has focused on the unsupervised learning of the feature space in which to perform clustering.

A notion of *distance* or *dissimilarity* is central to data clustering algorithms. Distance, in turn, relies on representing the data in a feature space. The k-means clustering algorithm (MacQueen et al., 1967), for example, uses the Euclidean distance between points in a given feature space, which for images might be raw pixels or gradient-orientation histograms. The choice of feature

space is customarily left as an application-specific detail for the end-user to determine. Yet it is clear that the choice of feature space is crucial; for all but the simplest image datasets, clustering with Euclidean distance on raw pixels is completely ineffective. In this technical review, we revisit cluster analysis and ask: *Can we use a data driven approach to solve for the feature space and cluster memberships jointly?*

We take inspiration from recent work on deep learning for computer vision (Krizhevsky et al., 2012; Girshick et al., 2014; Zeiler & Fergus, 2014; Long et al., 2014), where clear gains on benchmark tasks have resulted from learn- ing better features. These improvements, however, were obtained with *supervised* learning, whereas our goal is *unsupervised* data clustering. To this end, we define a parameterized non-linear mapping from the data space X to a lower-dimensional feature space Z, where we optimise a clustering objective. Unlike previous work, which operates on the data space or a shallow linear embedded space, we use stochastic gradient descent (SGD) via backpropagation on a clustering objective to learn the mapping, which is parameterized by a deep neural network. We refer to this clustering algorithm as *Deep Embedded Clustering*, or DEC.

Optimizing DEC is challenging. We want to simultaneously solve for cluster assignment and the underlying fea- ture representation. However, unlike in supervised learning, we cannot train our deep network with labeled data. Instead we propose to iteratively refine clusters with an auxiliary target distribution derived from the current soft cluster assignment. This process gradually improves the clustering as well as the feature representation.

Our experiments show significant improvements over state-of-the-art clustering methods in terms of both accuracy and running time on image and textual datasets. We evaluate DEC on MNIST comparing it with standard and state-of-the-art clustering methods (Nie et al., 2011; Yang et al., 2010). In addition, the experiments show that DEC is significantly less sensitive to the choice of hyperparameters compared to state-of-the-art methods. This robustness is an important property of the clustering algorithm since, when applied to real data, supervision is not available for hyperparameter cross-validation.

The proposed algorithm (DEC) clusters data by *simultaneously* learning a set of k cluster centers $\{\mu j \in Z\}kj=1$ in the feature space Z and the parameters θ of the DNN that maps data points into Z. DEC has two phases: (1) parameter ini- tialization with a deep autoencoder (Vincent et al., 2010) and (2) parameter optimization (i.e., clustering), where we iterate between computing an

auxiliary target distribution and minimizing the Kullback–Leibler (KL) divergence to it. We start by describing phase (2) parameter optimiza- tion/clustering, given an initial estimate of θ and {μj }kj =1 .

4.4.1 Clustering with KL Divergence

Given an initial estimate of the non-linear mapping fθ and the initial cluster centroids $\{\mu j\}_{j=1}^{k}$, the proposal is to improve the clustering using an algorithm that uses an unsupervised approach to improve the clustering of embedded points and cluster centroids. The algorithm involves learning from high confidence assignments using an auxiliary target distribution. This process is repeated until a convergence criterion is met.

The 2 steps being i) Soft Assignment (ii) KL Divergence Minimisation.

4.4.1.1 Soft Assignment

Following van der Maaten & Hinton (2008) we use the Student's t-distribution as a kernel to measure the similarity between embedded point z_i and centroid μ_j :

$$q_{ij} = \frac{(1 + \|z_i - \mu_j\|^2/\alpha)^{-\frac{\alpha+1}{2}}}{\sum_{j'}(1 + \|z_i - \mu_{j'}\|^2/\alpha)^{-\frac{\alpha+1}{2}}}, \qquad (1)$$

where $z_i = f_\theta(x_i) \in Z$ corresponds to $x_i \in X$ after embedding, α are the degrees of freedom of the Student's t distribution and q_{ij} can be interpreted as the probability of assigning sample i to cluster j (i.e., a soft assignment). Since we cannot cross-validate α on a validation set in the unsupervised setting, and learning it is superfluous (van der Maaten, 2009), we let α = 1 for all experiments.

4.4.1.2 KL Divergence Minimisation

$$L = \mathrm{KL}(P\|Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}. \qquad (2)$$

The choice of target distributions P is crucial for DEC's performance. A naive approach would be setting each pi to a delta distribution (to the nearest centroid) for data points above a confidence threshold and ignoring the rest. However, because qi are soft assignments, it is more natural and flexible to use softer probabilistic targets. Specifically, we would like our target distribution to have the following properties: (1) strengthen predictions (i.e., improve cluster purity), (2) put more emphasis on data points assigned with high confidence, and (3) normalize loss contribution of each centroid to prevent large clusters from distorting the hidden feature space. We compute pi by first raising qi to the second power and then normalizing by frequency per cluster:

$$p_{ij} = \frac{q_{ij}^2 / f_j}{\sum_{j'} q_{ij'}^2 / f_{j'}}, \qquad (3)$$

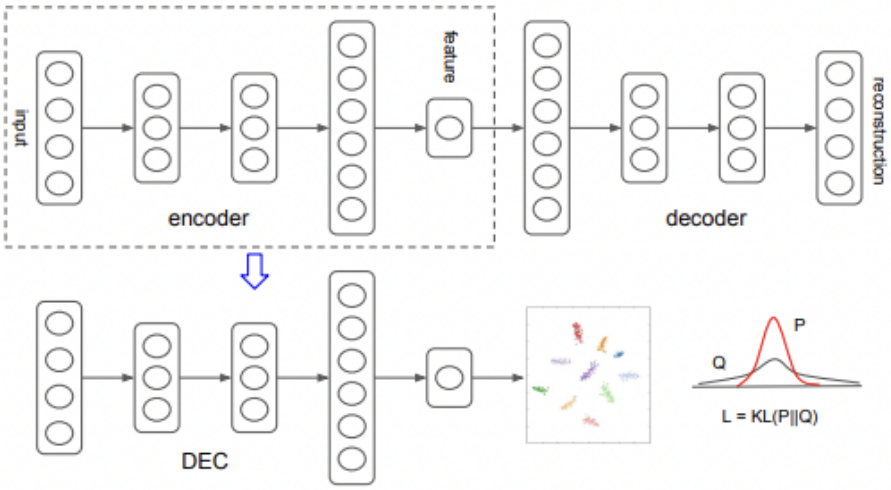


Fig 16: Network structure

4.4.2 Evaluation Metric

$$ACC = \max_{m} \frac{\sum_{i=1}^{n} \mathbf{1}\{l_i = m(c_i)\}}{n}, \qquad (10)$$

where l_i is the ground-truth label, c_i is the cluster assignment produced by the algorithm, and m ranges over all possible one-to-one mappings between clusters and labels.

4.4.3 Experiment Results

After reaching the tolerance threshold, our model converges at the 7280th iteration and we get an accuracy score of 0.86893, NMI score of 0.84168, ARI score of 0.80805 with a loss value = 0.1017.

```
Iter 7280: acc = 0.86893, nmi = 0.84168, ari = 0.80805  ; loss= 0.1017
delta_label  0.0009571428571428571 < tol  0.001
Reached tolerance threshold. Stopping training.
saving model to: results/DEC_model_final.h5
/usr/local/lib/python3.7/dist-packages/sklearn/utils/linear_assignment_.py
  FutureWarning)
acc: 0.8689285714285714
clustering time:  293.4301424026489
```

Fig 17: Terminal Output for DEC

The table below summarizes the same:

Metric	**Score**
Accuracy	0.86893
NMI	0.84168
ARI	0.80805

Table 7: Metric Score Results for DEC

4.4.3.1 Optimal Number of Clusters

So far we have assumed that the number of natural clusters is given to simplify comparison between algorithms. However, in practice this quantity is often unknown. Therefore a method for determining the optimal number of clusters is needed. To this end, we define two metrics: (1) the standard metric, Normalized Mutual Information (NMI), for evaluating clustering results with different cluster number:

$$NMI(l,c) = \frac{I(l,c)}{\frac{1}{2}[H(l) + H(c)]},$$

where I is the mutual information metric and H is entropy, and (2) generalizability (G) which is defined as the ratio between training and validation loss:

$$G = \frac{L_{train}}{L_{validation}}.$$

G is small when training loss is lower than validation loss,which indicate a high degree of overfitting. Fig. 16 shows a sharp drop in generalizability when cluster number increases from 9 to 10, which suggests that 9 is the optimal number of clusters. We indeed observe the highest NMI score at 9, which demonstrates that generalizability is a good metric for selecting cluster number. NMI is highest at 9 instead 10 because 9 and 4 are similar in writing and DEC thinks that they should form a single cluster.

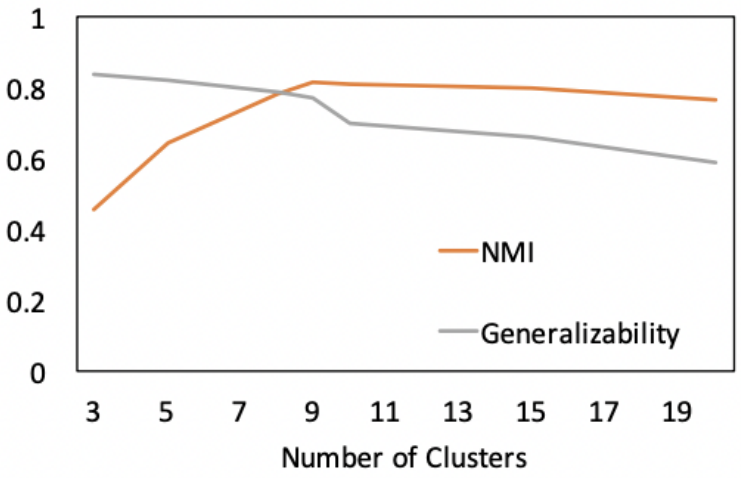


Fig 18: Selection of the centroid count, k. This is a plot of Normalized Mutual Information (NMI) and Generalizability vs. number of clusters.

4.4.4 Strengths and Weaknesses

Strengths	Weaknesses
- High Dimensionality	- Hyperparamters
- End to end framework	- Lack of interpretability
- Scalability	- Lack of theoretical framework

4.5 Density Based Spatial Clustering (DBSCAN)

A cluster in data space is defined as a continuous area of high point density, separated from other similar clusters by contiguous regions of low point density. Density-Based Clustering refers to unsupervised learning approaches that discover unique groups/clusters in the data based on this theory.

The basic method for density-based clustering is called Density-Based Spatial Clustering of Applications with Noise (DBSCAN). From a big quantity of data that contains noise and outliers, it may identify clusters of various sizes and forms.

Two parameters are used by the DBSCAN algorithm (scikit-learn, 2017):

- Eps: distance unit is used to identify the points that are close to a given location.

- minPts: The bare minimum of points (a threshold) need to cluster together in order to classify an area as dense.

DBSCAN visits each point of the database. DBSCAN executes exactly one such query for each point, and if an indexing structure is used that executes a neighborhood query in O(log n), and it has an overall average runtime complexity of O(n log n). The worst case run time complexity remains O(n^2).

With the help of research paper, “How to master the popular DBSCAN clustering algorithm for Machine Learning” by Abhishek Sharma, an overview of DBSCAN was understood and applied. According to the paper, DBSCAN groups ‘densely grouped’ data points into a single cluster. It can identify clusters in large spatial datasets by looking at the local density of the data points. The most exciting feature of DBSCAN clustering is that it is robust to outliers. It also does not require the number of clusters to be told beforehand, unlike K-Means, where we have to specify the number of centroids.

There also exists a much better and recent version of this algorithm known as HDBSCAN which uses Hierarchical Clustering combined with regular DBSCAN. It is much faster and accurate than DBSCAN.

4.5.1 Experiments with DBSCAN using IRIS dataset

Experimenting epsilon parameter

Epsilon	**0.01**	**0.05**	**0.1**	**0.5**	**1**	**5**
Number of Clusters	2	2	5	7	2	1
Accuracy	0.347	0.347	0.36	0.653	0.667	0.333
Silhouette Score	-0.209	-0.209	-0.533	0.191	0.687	-
Adjusted Random Score	0.0004	0.0004	0.003	0.523	0.568	-
Adjusted Mutual Info Score	0.011	0.011	0.039	0.611	0.732	-
Davies Bouldin Score	1.24	1.24	0.94	5.85	0.38	-
Calinski Harabasz Score	1.07	1.07	2.77	110.47	502.82	-

Table 8: Results of metrics for varying epsilon value

From the results of Table 8, we are able to see that the epsilon value of 1 gives the highest accuracy score while the epsilon values of 0.01 and 0.05 gives the same accuracy score.

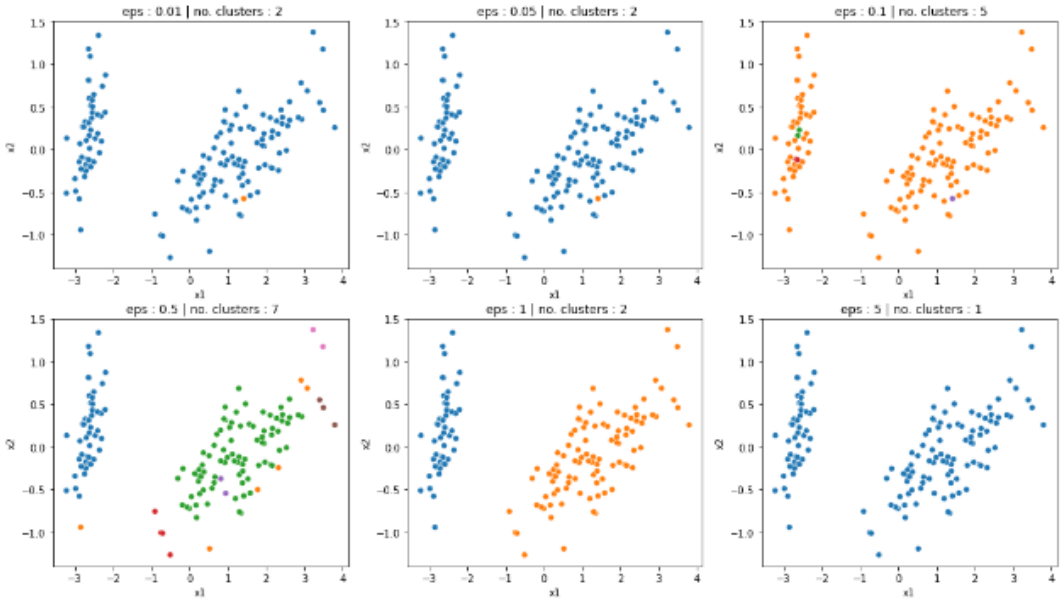


Fig 19: Plotting outputs of clusters for varying epsilon value

Experimenting min_sample parameter

min_sample	1	2	3	4	5	6
Number of Clusters	2	2	2	2	2	2
Accuracy	0.667	0.667	0.667	0.667	0.667	0.667
Silhouette Score	0.687	0.687	0.687	0.687	0.687	-
Adjusted Random Score	0.568	0.568	0.568	0.568	0.568	-
Adjusted Mutual Info Score	0.732	0.732	0.732	0.732	0.732	-
Davies Bouldin Score	0.383	0.383	0.383	0.383	0.383	-
Calinski Harabasz Score	502.822	502.822	502.822	502.822	502.822	-

Table 9: Results of metrics for varying min_sample value

From the results of Table 9, we can know that changing the min_sample value will not affect the metrics score and all will return the same values. Thus, all will return the same cluster plot shown in Fig 19.

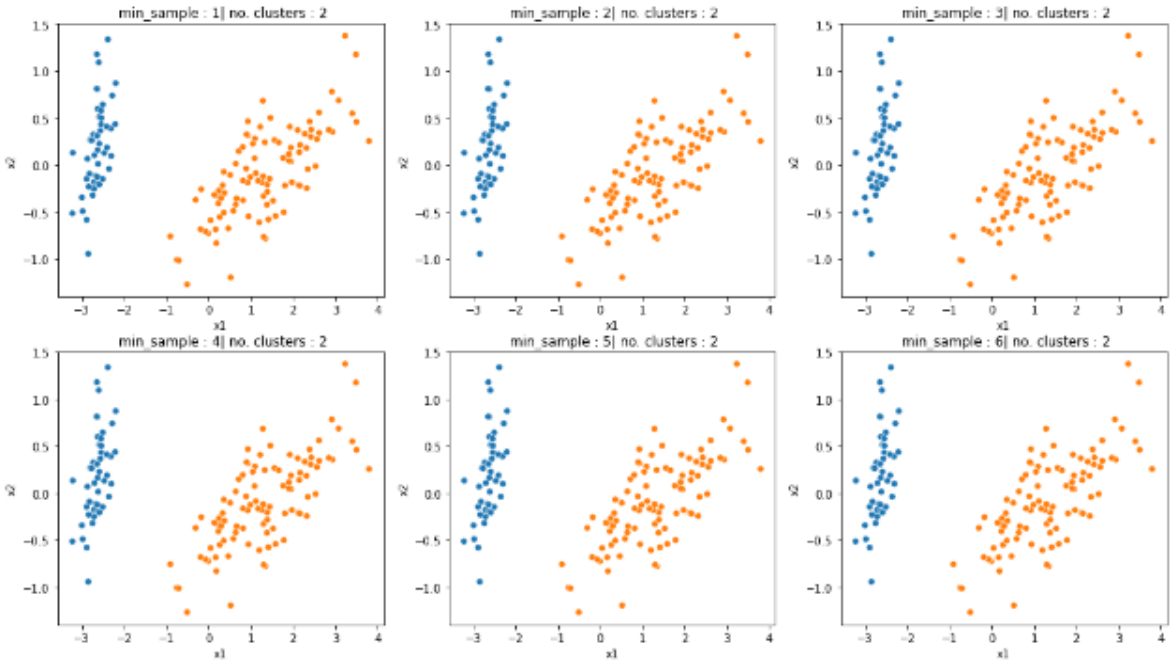


Fig 19: Plotting outputs of clusters for varying min_sample value

Determining the best epsilon value using K-Distance Graph

To use the K-Distance Graph method, we calculate the average distance between each point and its k nearest neighbors, where k = the MinPts value you selected. The average k-distances are then plotted in ascending order. From the K-Distance graph plotted, we can find the optimal Epsilon value at the point where the gradient is the highest.

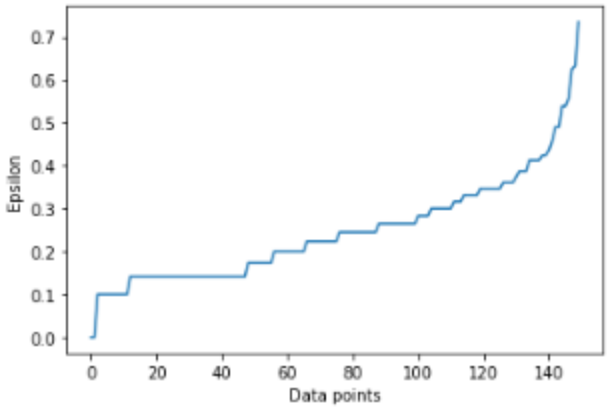


Fig 20: K-Distance graph for IRIS dataset

From Fig 20, we can know that the optimal epsilon value will be 0.4. Thus, we will be using this epsilon value and the min_sample value of 2 to implement our DBSCAN.

Metric	Score
Accuracy Score	0.647
Silhouette Score	0.242
Adjusted Random Score	0.705
Adjusted Mutual Info Score	0.692
Davies Bouldin Score	2.559
Calinsku Harabasz Score	79.797

Table 10: DBSCAN metrics score for Iris Dataset

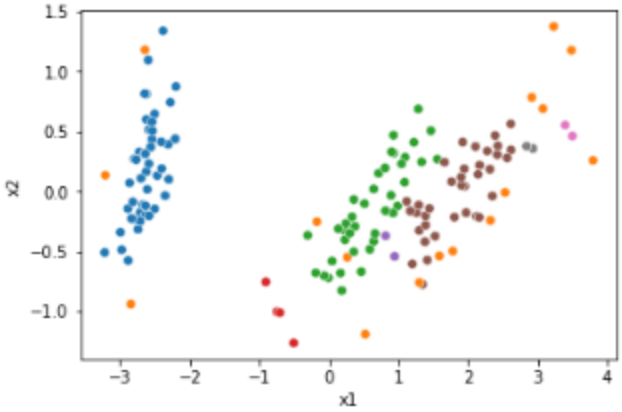


Fig 21: Plotting of clusters based on K-Distance Graph epsilon value for Iris Dataset

From the values generated in Table 10, we can know that DBSCAN do not really work well with the IRIS dataset with the metrics score generated. More clusters, in this case 8 clusters, are also being formed due to the small epsilon value. To visualize the true IRIS classes, we used PCA Decomposition to help us generate.

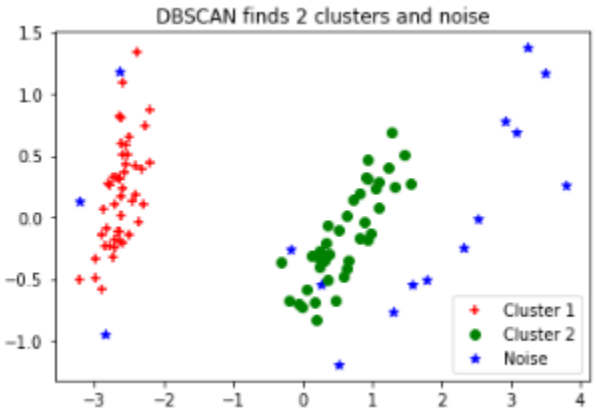


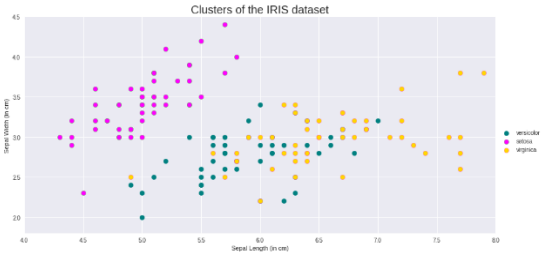
Fig 22: Plotting of clusters using PCA Decomposition for Iris Dataset

Hence, we can see that DBSCAN produced 3 groups but it resembles a two-cluster solution. This is because DBSCAN is a two-cluster solution where the clusters are represented by values 0 and 1 while -1 refers to the outliers (noise). From this, we can see that DBSCAN did not produce the best outcome which is the actual clusters (species) of the dataset. IRIS dataset did not bring out the most powerful features of DBSCAN - noise detection and capability to form clusters of arbitrary shapes.

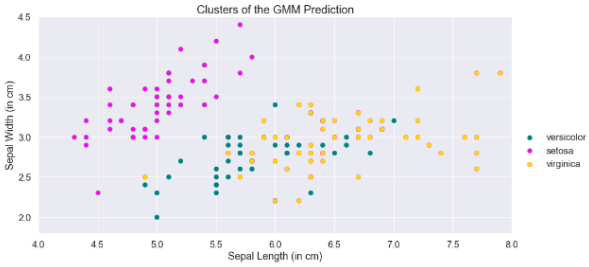
4.5.2 Strengths and Weaknesses of DBSCAN

Strengths	Weaknesses
- Do not require pre-specification of number of clusters - Ability to find arbitrarily sized and shaped clusters - Ability to identify noise data while clustering	- Does not work well in high dimensional data - Does not work well in case of varying density clusters - Difficult to determine the epsilon value

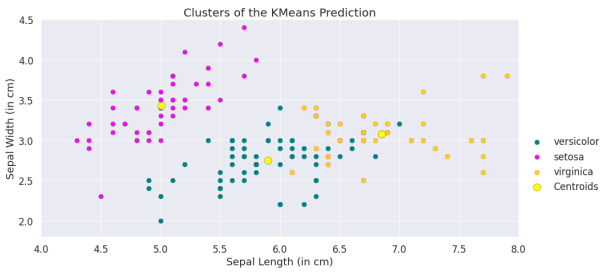
5. Conclusion



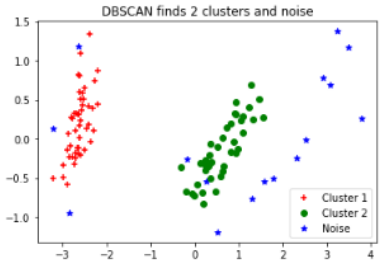
IRIS dataset



GMM Clustering



K-Means Clustering



DBSCAN Clustering

Overall, the 5 unsupervised clustering algorithms performs differently in how they do the clustering. K-Means and GMM are different where K-Means only account for the mean to update centroid while GMM accounts for both the mean and the variance which makes it a more accurate

model. We also learnt that DBSCAN is a density-based clustering while K-Means is centroid-based clustering. However, from the results and outputs generated using IRIS dataset, we can see that DBSCAN does not work well as it will always produce a 2-clusters output which differs from IRIS dataset (3 clusters). The IRIS dataset serves as a good example to show which algorithms works best. Amongst the 3 algorithms performed with IRIS dataset, GMM performs the best with an outstanding accuracy score of 0.967.

	K-Means	GMM	DBSCAN
Accuracy Score	0.893	0.967	0.647
Silhouette Score	0.735	0.650	0.242
Adjusted Random Score	0.730	0.904	0.705
Adjusted Mutual Info Score	0.755	0.898	0.692
Davies Bouldin Score	0.661	0.748	2.559
Calinsku Harabasz Score	561.627	481.781	79.797

Table 11: Overview of metrics score for the algorithms performed on IRIS dataset

Meanwhile, DEC works by iteratively optimizing a KL divergence based clustering objective with a self- training target distribution. This method can be viewed as an unsupervised extension of semi supervised self-training. This framework provides a way to learn a representation specialized for clustering without ground truth cluster membership labels.

Empirical studies demonstrate the strength of our proposed algorithm. DEC offers improved performance as well as robustness with respect to hyperparameter settings, which is particularly important in unsupervised tasks since cross-validation is not possible. DEC also has the virtue of linear complexity in the number of data points which allows it to scale to large datasets.

Agglomerative clustering divides the population into several clusters such that data points in the same clusters are more similar and data points in different clusters are dissimilar. It uses a

bottom-up approach, initially, each datapoint is a cluster of its own, further pairs of clusters are merged as one moves up the hierarchy.

The library Clustimage is used to cluster and evaluate the clustering and as we can see from the table below, the clustering is evaluated on the basis of silhouette score. Agglomerative clustering gave an average silhouette score of 0.576 and estimated 10 clusters, which is approximately one for each digit from 0-9 in the MNIST dataset.

Both DEC and Agglomerative are unsupervised clustering methods and give approximately same number of clusters (≈ 10). From the table below, we see that K-means performs poorly when it takes the number of clusters to be 10 as compared to DEC. A potential reason could be the different metrics used to calculate the distance between the data points as K-means uses metrics like euclidean distance (or cosine similarity) whereas DEC is composed of a neural network model, uses autoencoders and mutual information is calculated using KL Divergence method.

Clustering method	**Evaluation Method**	**Evaluation Score**
K-Means Clustering	Accuracy	0.564
	No of clusters	10
Agglomerative Clustering	Silhouette Score	0.576
	No of clusters	10
Deep Embedded Clustering	Accuracy	0.8689
	NMI Score	0.84168

Table 12: Overview of metrics score for the algorithms performed on MNIST dataset

6. References

1) Aggarwal, Charu C and Reddy, Chandan K. Data clustering: algorithms and applications. CRC Press, 2013.
2) Alelyani, Salem, Tang, Jiliang, and Liu, Huan. Feature selection for clustering: A review. Data Clustering: Algorithms and Applications, 2013.
3) Boutsidis, Christos, Drineas, Petros, and Mahoney, Michael W. Unsupervised feature selection for the kmeans clustering problem. In NIPS, 2009.
4) Chouinard J-C. 2022 May 5. What is KMeans Clustering Algorithm (with Example) – Python. JC Chouinard. https://www.jcchouinard.com/kmeans/.
5) Code Studio. wwwcodingninjascom. https://www.codingninjas.com/codestudio/library/mini-batch-kmeans.
6) De la Torre, Fernando and Kanade, Takeo. Discriminative cluster analysis. In ICML, 2006.
7) Elkan C. 2003. Using the Triangle Inequality to Accelerate -Means.
8) Girshick, Ross, Donahue, Jeff, Darrell, Trevor, and Malik, Jitendra. Rich feature hierarchies for accurate object detection and semantic segmentation. In CVPR, 2014.
9) Halkidi, Maria, Batistakis, Yannis, and Vazirgiannis, Michalis. On clustering validation techniques. Journal of Intelligent Information Systems, 2001.
10) Kang C. 2020 Oct 26. K-Means Clustering for Imagery Analysis. Chan`s Jupyter. https://goodboychan.github.io/python/machine_learning/natural_language_processing/vision/2020/10/26/01-K-Means-Clustering-for-Imagery-Analysis.html.
11) Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In NIPS, 2012.
12) Li, Tao, Ma, Sheng, and Ogihara, Mitsunori. Entropybased criterion in categorical clustering. In ICML, 2004
13) Liu, Huan and Yu, Lei. Toward integrating feature selection algorithms for classification and clustering. IEEE Transactions on Knowledge and Data Engineering, 2005.
14) Long, Jonathan, Shelhamer, Evan, and Darrell, Trevor. Fully convolutional networks for semantic segmentation. arXiv preprint arXiv:1411.4038, 2014.
15) MacQueen, James et al. Some methods for classification and analysis of multivariate observations. In Proceedings of the fifth Berkeley symposium on mathematical statistics and probability, pp. 281–297, 1967.

16) Medewar S. 2022 May 13. Code Studio. wwwcodingninjascom. [accessed 2022 Oct 16]. https://www.codingninjas.com/codestudio/library/mini-batch-kmeans.
17) Nie, Feiping, Zeng, Zinan, Tsang, Ivor W, Xu, Dong, and Zhang, Changshui. Spectral embedded clustering: A framework for in-sample and out-of-sample spectral clustering. IEEE Transactions on Neural Networks, 2011.
18) Olcay Akman and Josselyn Gonzales: Data Clustering and Self-Organizing Maps in Biology. Algebraic and Combinatorial Computational Biology, 2019.
19) Pooh ® IS. 2018 Nov 3. 2.3.9.3. Homogeneity, Completeness and V-measure. 최현웅의 항공IT.https://esigma6.wordpress.com/2018/11/03/2-3-9-3-homogeneity-completeness-and-v-measure/.
20) scikit-learn. API Reference — scikit-learn 0.24.2 documentation. scikit-learnorg. [accessed 2022 Oct 16]. https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics.
21) scikit-learn. 2017. sklearn.cluster.DBSCAN — scikit-learn 0.22 documentation. Scikit-learnorg. https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html.
22) Taskesen E. 2020. Gridsearch — clustimage clustimage documentation. Githubio. [accessed 2022 Oct 16]. https://erdogant.github.io/clustimage/pages/html/Cluster%20Evaluation.html#silhouette.
23) Von Luxburg, Ulrike. A tutorial on spectral clustering. Statistics and computing, 2007.
24) Vincent, Pascal, Larochelle, Hugo, Lajoie, Isabelle, Bengio, Yoshua, and Manzagol, Pierre-Antoine. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. JMLR, 2010.
25) Xiang, Shiming, Nie, Feiping, and Zhang, Changshui. Learning a mahalanobis distance metric for data clustering and classification. Pattern Recognition, 2008.
26) Xie J, Girshick R, Farhadi A. Unsupervised Deep Embedding for Clustering Analysis, 2016.
27) Xing, Eric P, Jordan, Michael I, Russell, Stuart, and Ng, Andrew Y. Distance metric learning with application to clustering with side-information. In NIPS, 2002.
28) Yang, Yi, Xu, Dong, Nie, Feiping, Yan, Shuicheng, and Zhuang, Yueting. Image clustering using local discriminant models and global integration. IEEE Transactions on Image Processing, 2010.
29) Zeiler, Matthew D and Fergus, Rob. Visualizing and understanding convolutional networks. In ECCV. 2014.