MATH572: Project 15- Comparison of Clustering Algorithms

1 Abstract

In this data-driven world, it is vital to learn how to utilize the huge amounts of data available for extracting meaningful insights. Powerful Artificial Intelligence-based techniques could be implemented to group data points with other data points similar in characteristics and inherent structure to discover trends and characteristics that could sow the seeds for further research and analysis [1].

Clustering is a machine learning technique that groups data into 'clusters' where points in a cluster have similar characteristics which differentiate them from points in the other clusters. This project aims to run various clustering algorithms on a (459,2207) dimensional dataset and compare the functioning of the clustering techniques along with testing two types of dimensionality reduction techniques.

The clustering algorithms evaluated are k-means clustering, single-linkage clustering, Gaussian mixture model with expectation-maximization, density-based spatial clustering of applications with noise (DBSCAN), and balance iterative reducing and clustering using hierarchies (BIRCH). Dimensionality reduction techniques such as principal component analysis and t-distributed stochastic neighbor embedding (t-SNE) were utilized and tested. The evaluation involves comparing cluster assignments and visualizations. The objective is to gain a comprehensive understanding of the advantages, limitations, and working of the clustering and dimensionality reduction algorithms, and their suitability for the given data.

2 Introduction

Artificial Intelligence refers to the mimicking of human intelligence in terms of learning and thinking by machines with the help of algorithms. Machine Learning is a subset of Artificial Intelligence that involves computers learning by recognizing patterns from the data and being able to make predictions without explicit programming [2]. Machine learning techniques are divided into supervised learning, unsupervised learning, and reinforcement learning techniques. In brief, supervised learning involves machines understanding

patterns from data and their labels, unsupervised learning involves machines recognizing patterns in data without labels, and finally, reinforcement learning refers to reward-based learning by an agent from its mistakes and experiences during its interactions with its surroundings [2].

Clustering is an unsupervised machine learning technique that aims to group data into clusters based on the similarities in their features. Data points in a cluster are similar in properties which are different from the ones in another cluster. This aids in associating certain characteristics with certain groups of data. Several clustering algorithms have been developed that work better with data of different feature combinations.

Among the many clustering algorithms, this project compares k-means clustering, single-linkage clustering, Gaussian mixture model with expectation-maximization, DBSCAN, and BIRCH. The clustering algorithm has been tested on a dataset with 2207 features which is a high-dimensional dataset. This could pose issues in terms of the processing time and resources needed, as well as cause overfitting to the training data that is, high performance on the data the model is trained on but weaker generalization to the unseen data points [3]. Data points not belonging to a particular cluster could be recognized as genuine patterns and not random fluctuations or noise. Dimensionality reduction techniques such as principal component analysis and t-distributed stochastic neighbor embedding (t-SNE) were utilized to reduce the number of features and retain vital ones. The algorithms were run in varying combinations and evaluated and visualized to interpret the results. The report explains the clustering algorithms tested and highlights their differences to explain the suitability of each of them to the given data.

2.1 Background

This section encompasses the necessary information regarding the background of the techniques and algorithms used in this project.

2.1.1 Clustering

Clustering is an unsupervised machine learning algorithm that involves the learning of patterns and relationships in the data without labels using suitable algorithms. It is specifically suitable when data and the technique to be used are completely unknown however, the researcher is left oblivious to the hidden

patterns in the data which would not be the case if labels were present and briefly examined beforehand. As the name suggests, clusters of data points are created based on their properties and features. Data points in different groups largely vary from those in the other clusters or groups. Clustering or cluster analysis involves initializing the number of clusters to divide the data into those number groups. It automates the process of making pairwise comparisons which would be inconvenient if done manually. Figure 1 is a visual representation of the grouping of the data points into 3 clusters, denoted by the three colours used. It is to be noted that data points in a cluster are closer to each other and the center of the cluster they belong to than to those of other clusters.

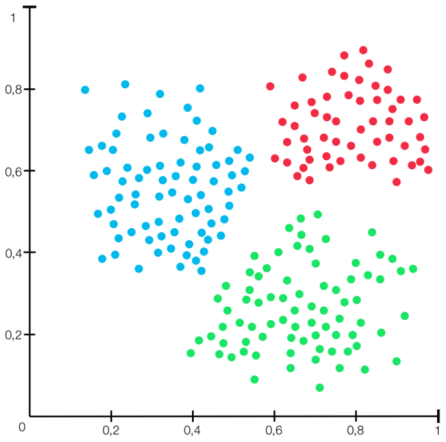

Figure 1: Data points in three clusters [4]

Clustering is widely used in data analysis and machine learning projects to discover the natural intricate patterns and inherent structure of the data points that could indicate grouping between them. Popular applications include anomaly detection, image segmentation, customer segmentation, feature engineering, and medical imaging among others. There are various clustering algorithms to choose from that suit different datasets so in this project, a few clustering algorithms have been experimented with to determine which of those with what setting would suit the given dataset. Clustering algorithms have been imported from the scikit-learn [5] and the SciPy [6] libraries in Python that house various machine-learning and scientific algorithms' implementations respectively.

Clustering is of two types- hard clustering and soft clustering. Hard clustering algorithms are those in which a given data point completely belongs to one

cluster or does not. Soft clustering algorithms assign to each data point a probability or likelihood score associated with belonging to the pre-determined clusters [7]. Clustering algorithms are of several types such as centroid-based, density-based, hierarchical/connectivity clustering, and distribution-based clustering.

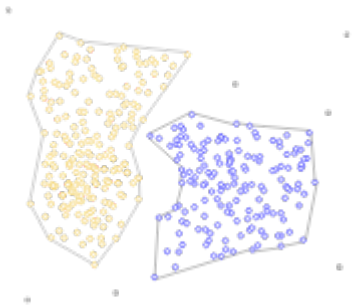

Figure 2: Density-based Clustering [8]

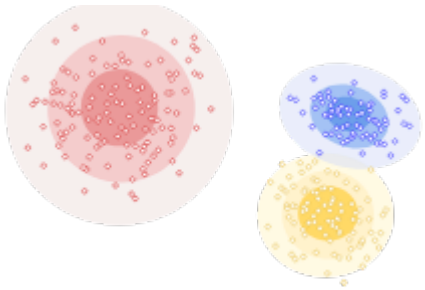

Figure 3: Distribution-based Clustering [8]

Centroid-based clustering represents each cluster with a single centroid or mean vector to which each data point is compared in terms of its proximity to the centroids. Density-based models (Figure 2) form clusters based on the density of the data points in the data space. Connectivity models suggest clusters based on distance, closer points tend to be similar. They are either partitioned into clusters from a single big cluster or aggregated into clusters when each data point is assigned a cluster of its own initially. Finally, distribution models (Figure 3) group data in terms of their probability or likelihood of being a part of a distribution and thus the cluster representing that cluster [9, 10].

2.1.2 k-Means Clustering

A widely used clustering technique is k-means clustering which has been used several times throughout this project. This technique involves clustering the data vectors into k different clusters based on their distance to the cluster means or centroids generated to minimize the variance within each cluster [11]. These centroids are not part of the original dataset that is being organized. Clusters are assumed to be of the same sizes and to be spherical. It can be classified as a hard clustering and centroid-based algorithm.

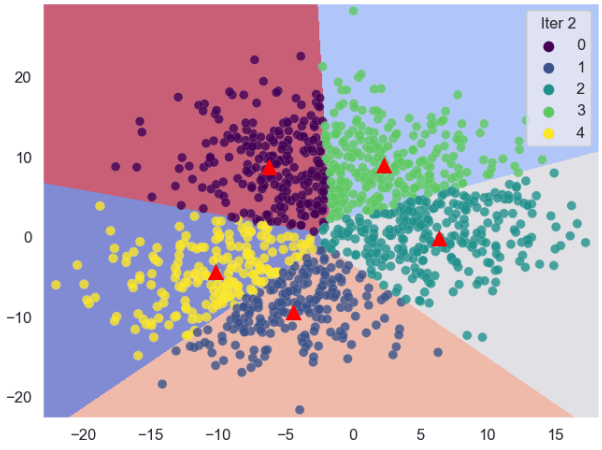

Figure 4: Voronoi Diagram: k-means clustering [12]

A Voronoi diagram is created out of the vector space as displayed in Figure 4. It can also be noticed that convex clusters are generated due to the splitting along straight lines. The different colours correspond to the different clusters while the red 'X's correspond to the cluster centres. This is known to be a simple but efficient technique to implement which explains its popularity.

2.1.3 Single Linkage Clustering

It is a hierarchical or agglomerative clustering algorithm that focuses on joining nearby clusters iteratively to reduce the number of clusters from as many as the data points to the pre-determined number (harikamed). It assigns one cluster to each data point making it a hard-clustering algorithm. A unique attribute of these algorithms is the ability to create a dendrogram that visualizes the hierarchical connections between the data points. It is a plot of the observations/clusters and the distance between the clusters. The distance between pairs of observations is calculated in terms of the linkage criteria. For this particular algorithm, as given by the name, the single linkage criterion is used where the distance between the two clusters is calculated as the minimum distance between a pair of data points belonging to each of the two clusters and it is given by $d(u, v) = min(u[I], v[j])$ where u and v are clusters and i,j correspond to the indices of the elements in the two clusters respectively [13].

2.1.4 Expectation-Maximization Algorithm

The expectation-maximization Algorithm is a probabilistic or soft clustering algorithm where each point belongs to every cluster with a certain probability. Its functioning is similar to that of k-means but the soft clustering or probabilistic cluster assignment to vectors and consideration of both mean and variance instead of just mean makes it different. It can be utilized to unravel the properties of a distribution that at least some data points are assumed to be drawn from.

Gaussian Mixture Model (GMM) assumes its data points are generated from a mix of Gaussian or normal distributions which represent each of the clusters and it is a probabilistic model. They are extremely helpful for overlapping distributions and non-circular clusters. The distributions' parameters must be discovered to recognize the clusters and the probabilities [9]. Gaussian or normal distributions are bell-shaped curves with equal mean, median, and mode. The diagram (Figure 5) highlights a point in red and the associated probabilities for that point to be in each of the clusters are displayed. The probability that the point is a part of cluster green is 0, the probability of belonging to blue is 0.2 and that of being in cyan is 0.8.

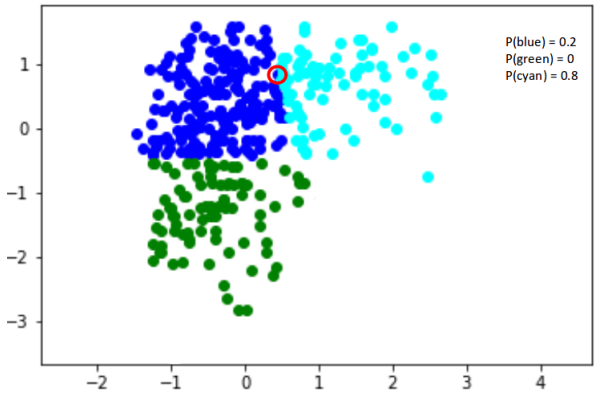

Figure 5: GMM [9]

A Gaussian mixture model is comprised of the mean/center (μ), covariance/width (σ), and mixing probability (relative weight of each Gaussian in the mixture) [10]. To find the parameters for the Gaussian models, the Expectation-Maximization algorithm can be utilized. The expectation step involves calculating the probability a data point belongs to one of the clusters representing each of the distributions and the minimization step updates the

parameters- mean, variance, and density which are then used to calculate probabilities for each of the data points [7].

Expectation involves the computation of the posterior probabilities using the probability density function (Figure 6) of each data point belonging to each of the clusters (Figure 7).

$$f(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Figure 6: Probability density function of a Gaussian distribution with one dimension/feature [9]

where x is the input vector,μ is the 2D mean vector, and σ is the 2×2 covariance matrix

$$r_{ic} = \frac{\text{Probability Xi belongs to c}}{\text{Sum of probability Xi belongs to } c_1, c_2, .. c_k} = \frac{\pi_c \mathcal{N}(x_i \ ; \ \mu_c, \Sigma_c)}{\sum_{c'} \pi_{c'} \mathcal{N}(x_i \ ; \ \mu_{c'}, \Sigma_{c'})}$$

Figure 7: Formula of the probability that x_i belongs to cluster c1, c2, ... ck [9]

Maximization of the likelihood of the data by the update of the parameters from the distributions calculated earlier along with the posterior probabilities is done next. The values of the other parameters were left at their default values. If the GMM consists of 2 Gaussian/normal distributions which are given by $N(\mu_1, (\sigma_1)^2) and N(\mu_2, (\sigma_2)^2))$, five parameters (collectively called θ have to be estimated- $\mu_1, (\sigma_1)^2)$, $\mu_2, (\sigma_2)^2)$, probability the data comes from the first distribution (p). The PDF of the GMM is given by [9]-
$g(x|\theta) = pg_(1)(x|\mu_1, (\sigma_1)^2) + (1-p)g_(2)(x|\mu_2, (\sigma_2)^2)$

2.1.5 DBSCAN

Density-based spatial clustering of applications (DBSCAN) is a density-based, soft clustering algorithm. The algorithm is capable of handling data with arbitrary shapes and of large sizes too and generates arbitrarily shaped clusters based on the number of points in a specific region (density). The number of

clusters and outliers in a dataset can be recognized as well [14], although the former need not be known and the latter is not an issue due to the algorithm's robustness. Clusters are assumed to be dense regions that are separated by less dense regions [14] as shown in Figure 8.

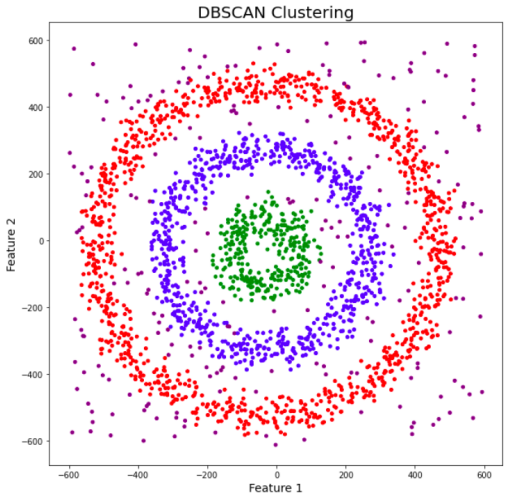

Figure 8: DBSCAN Clustering [14]

Clusters are made based on the number of data points that are defined to be in a cluster that is, the minimum number of data points for an area to be considered highly dense and, the distance (called epsilon) to consider a data point to be in a cluster with data points already present in it or the maximum distance for two points to be considered neighbours [15]. The algorithm is highly sensitive to these values such as a low epsilon value will create more clusters whereas a higher value will lead to smaller clusters disabling us from understanding the intricate low-level details. Core points have the minimum number of points configured around them, border points have a core point within epsilon distance, and points other than these are considered noise as seen in Figure 9.

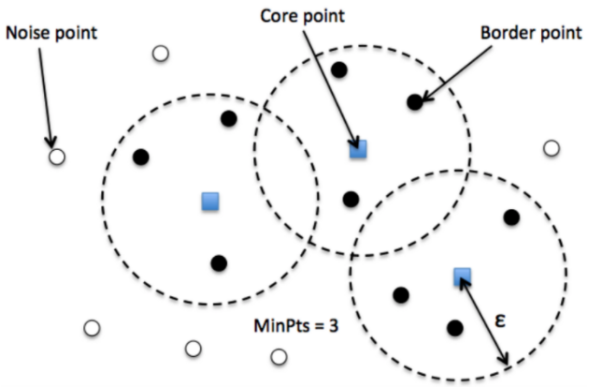

Figure 9: Classification of data points in DBSCAN [16]

2.1.6 BIRCH

Balance Iterative Reducing and Clustering using Hierarchies (BIRCH) is a hierarchical soft clustering algorithm that is said to be suited for large numerical datasets so categorical values cannot be handled simply. It also combines density-based clustering in the aspect of recognizing dense regions to make subclusters. The data is structured into summaries with the same distribution information as the data points [15]. This compact summary data structure is called a cluster feature tree and this makes the algorithm memory-efficient. The clustering takes place on the cluster feature tree generated. This algorithm is also referred to as two-step clustering as it involves the two steps of building the cluster tree and clustering.

The cluster tree is populated with summaries of the clusters or regions with many data points and each summary or entry is given by a clustering feature- a triple, "CF = (N, LS, SS), where N is the number of data points in the cluster, LS is the linear sum of the N data points, and SS is the squared sum of the N data points" [17]. A cluster feature tree consists of sub-clusters instead of data points individually. These entries are made in a leaf node or nodes with parents but no children of a tree. It can be described with two parameters, threshold- the maximum entries that can be made in a leaf node, and branching factor- maximum cluster features sub-clusters in each node that is not a leaf. Threshold and tree size are inversely proportional. It is also important to mention the number of clusters to not receive the intermediate clusters as the output. The clustering is made in one iteration and improved in the next few, unlike others that could be run for many iterations [17] such as in Figure 10.

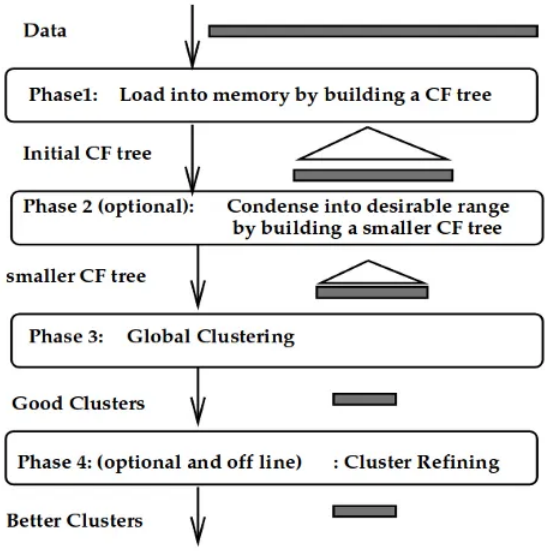

Figure 10: BIRCH algorithm [17]

2.1.7 Principal Component Analysis

To alleviate the problems of sparsity, indecipherable visualization, and overfitting caused by data of high dimensions or features, dimensionality reduction procedures such as PCA and t-SNE could be used to retain necessary features and discard redundant features. This would also help to get rid of unnecessary features in the presence of correlated features. Lesser dimensions could help in better data visualization and understanding the high-level patterns in the data. Principal Component Analysis (PCA) produces a new set of features or principal components which are linear combinations of the original set of features. The most important or major sources of variability in the dataset are captured in all directions to provide a global view of how the data points are distributed along all directions. The principal components are the data transformed into orthogonal axes and the eigenvector of the matrix of covariates between features. The first principal component is said to account for most variance in the data so it is said to be the most important [3]. It aims to retain the majority of the overall linear variance of the dataset.

2.1.8 t-SNE

t-Distributed Stochastic Neighbor Embedding (t-SNE) can be used as a dimensionality reduction technique and for visualizing high-dimensional datasets. It can be used to retain non-linear variance or local variance so would be useful in the case of non-linear data, unlike PCA. In brief, it is a non-linear technique that is concerned with random probability thus producing slightly varying results each time on a dataset (being stochastic/non-deterministic), while trying to retain the structure of its neighbouring points.

Stochastic behaviour can be controlled when run in Python. Based on weighted distance functions, the likelihood of one point selecting another as its neighbor is computed ignoring the distances between clusters or the cluster sizes. Similar data points are assigned higher probability than dissimilar ones from the probability distribution constructed. Kullback-Leibler divergence is defined as the expected value of the logarithmic difference between the probability distributions in the higher and lower dimensions. The probability distribution for higher dimensions is repeated for lower dimensions in an iterative manner until this difference is minimized [18].

Epsilon- the learning rate or rate of update of parameters, and perplexity which can be described as the number of neighbours desired for each point are the parameters of this technique. The number of iterations to run the algorithm has to be pre-determined too. It is highly sensitive to parameter values such as lower perplexity could lead to lesser clusters [18].

The Euclidean distance between the points is calculated and the joint probabilities explain the likelihood of two points being neighbours which is proportional to the distance between them. These joint probabilities are defined by the t-distribution, with parameters such as the mean, degrees of freedom, and scale (sigma) [19]. Gradient descent optimization is then applied to generate a low-dimensional embedding. The joint probabilities are attempted to be preserved while minimizing the Kullback-Leibler divergence between the original data and the created embedding.

2.2 Problem Description

The problem addressed in this project is the comparison of clustering algorithms on the given dataset. The dataset consists of observations with multiple features tabulated in a spreadsheet. The goal is to analyze the performance of the

clustering algorithms in terms of understanding the patterns from the given dataset by evaluating the clustering assignments a matrices, visualizing scatterplots, and three-dimensional plots. Specifically, the project aims to understand the ability of k-means clustering with the right number of clusters, on the original dataset as well as datasets reduced by following principal component analysis (PCA) and t-SNE. This is to understand how PCA affects the structure of the data and the consistency of cluster assignments.

The robustness of the k-means clustering has been assessed by checking the impact of different initializations for the cluster centroids on the resulting cluster assignments. Similarly, single-linkage clustering and Gaussian mixture model (GMM) with expectation-maximization algorithm were run to understand the differences in cluster shapes made by all these algorithms. For the GMM, the percentage of observations with multiple cluster assignments with a probability greater than a certain threshold was computed. DBSCAN and BIRCH algorithms were run on the original dataset and the reduced datasets. Silhouette score was calculated for all clustering trials and clusters made were visualized. The research seeks to learn more about the strengths, disadvantages, and characteristic features of each clustering technique by tackling their issues and evaluating the outcomes.

2.3 Data

The data has been downloaded from the website-'https://russellmilne.com/teaching/gvrd-rangestd.csv'. The data has been loaded into the program in Python using a function from the Pandas library [20], as a dataframe with only floating-point values. A dataframe is a tabular collection of data that aids in simple storage, retrieval, and understanding of the data. The dataset consists of numerical values solely with each column representing a feature and each row representing the value of that feature across observations, originally. The original dimensions of the data are (5, 459) which indicates 5 rows and 459 columns. However, contrary to the usual standards, each column in the dataset represents one observation and each row represents the value of a given feature across the observations. Therefore, the transpose of the data has been used further, changing the dimensions to (459, 2207) that is, there are 459 rows and 2207 columns. After transposing, the columns represent features and the rows represent the observations. There were no null or missing values in the dataset along with no duplicated observations so this simplified the analysis.

3 Experiment Details

The experiment of comparing various clustering algorithms in combination with dimension reduction techniques was carried out after ensuring the data had no missing or duplicate values to be suitable for the techniques planned to use on this data. The code is written in Python and can also be accessed from this link (https://colab.research.google.com/drive/1UAEFaeSWRPrzHxjYONvIWMV9KBBsna7x?usp=sharing) or from the Appendix (Section 8.2).

3.1 Principal Component Analysis

PCA was performed on the dataset to reduce the number of features from 2207 to only 3. On checking the percentage of explained variance of the three components, very low values of 18% and lower were observed using a scree plot (Figure 24). A scree plot is a line plot of the explained variance ratio for each principal component. After which, PCA was run without any configuration done and the same scree plot was made with 459 components ($min(nfeatures, nobservations)$). For 90% of the total variance to be accounted for, 180 principal components are needed as the ratio levels off after this value (Figure 11). However, the clustering algorithms' performance did not improve compared to what it was with 3 principal components. Therefore, PCA was initialized with 3 components, sticking to the project description.

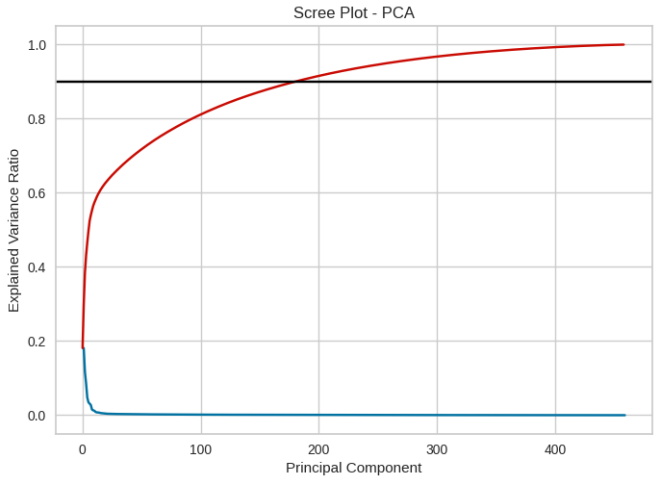

Figure 11: PCA cumulative variance at 90% threshold

3.2 Using t-SNE

t-SNE reduction was performed on the full-dimensional dataset to reduce the dimension to 2. All the other parameters were left at the default values including perplexity and the learning rate. Visualization of high-dimensional data is made possible in a lower-dimensional space while preserving the local structure and relationships between the data points, to a good extent. The tsne() function from the scikit-learn library has been used to implement the t-SNE algorithm. The number of dimensions to reduce to was set as 2, 2000 iterations were configured for better convergence, and the random seed was set to 0 for consistent results every time the code was run.

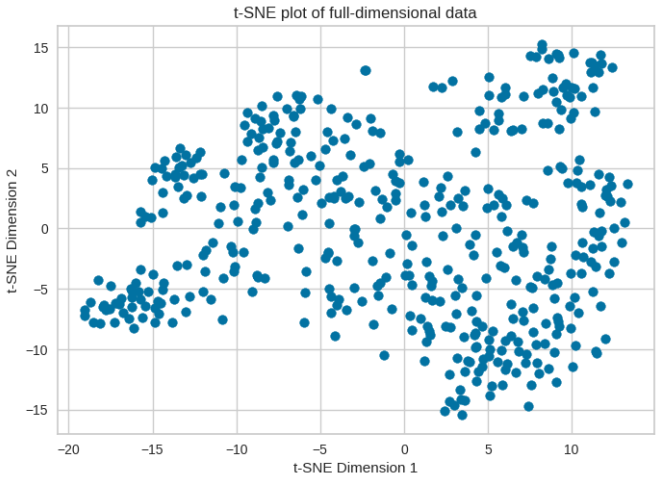

Figure 12: Final t-SNE visualization

A higher value of perplexity has been observed to produce a lower value of divergence from different settings of the perplexity from 5 to 50. This range was selected as it is commonly used in machine learning studies [18]. With the increase in the perplexity, the data is better separated appearing like clusters made by the other algorithms. The divergence values observed for the initial case with no perplexity set and the latter case with perplexity showcased a dip in the divergence as needed. The perplexity value was selected to be one of the values giving lesser divergence- 40 between 30 (the default value) and 50 (the maximum value). Thus, the divergence value dropped from 0.738 to 0.655 by tweaking the perplexity. While t-SNE is a visualization and reduction technique, data clusters can be observed which can be compared with the results of the other clustering algorithms. The final t-SNE visualization is given in Figure 12. The initial graph and the divergence versus perplexity graphs are given in the Appendix as Figures 25 and 26.

3.3 k-Means Clustering

k-means clustering was performed using scikit-learn library's implementation for good efficiency, speed, and ease of usage. Initially, k-means clustering was attempted with a random value of 5 for the number of clusters to form from the given data in an attempt to choose a number greater than 3. Along with this, the random state which is the initial seed for the cluster selection was chosen to be 0. The method of initialization which refers to the method of selecting the initial clusters was set to kmeans++ to select centroids efficiently. The number of runs to be made with different centroid seeds was set to 10 although the efficient kmeans++ was used instead of a random or custom initialization due to the high dimensionality of the data. The threshold for the difference in cluster centers between consecutive iterations indicating convergence is called tolerance was left to be the default along with the other parameters such as the iterations per run in the algorithm. Apart from the k values, the rest of the hyperparameters were set the same throughout the project.

The k-means algorithm to use was left as 'lloyd' which is the regular algorithm involving the assignment of each of the data points to their closest centroid based on the Euclidean distance between them and the update of the centroid of a cluster based on the mean of the data points in that cluster [21]. This is repeated until convergence that is, until the cluster assignments stop changing, or until the maximum number of runs is reached. In a 2-dimensional case, the Euclidean distance formula is given in Figure 13, this can be extended for more dimensions. $distance(x, y) = \sqrt{((x_2 - x_1)^2 + (y_2 - y_1)^2)}$ where $x = (x_1, x_2)$ and $y = (y_1, y_2)$.

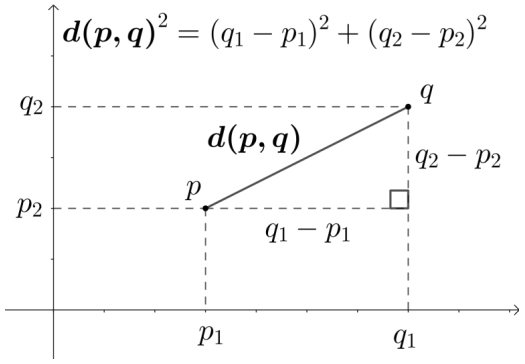

Figure 13: Euclidean distance for two points and two features

According to scikit-learn, the average amount of time this k-means

implementation takes to run as a function of the input size or the average time complexity is given by $O(knT)$, where n is the number of samples and T is the number of iterations indicating a linear increase in time with sample size. The worst-case time complexity of the algorithm is given by $O(n^(k + \frac{2}{p}))$ where n is the number of samples, p is the number of features, and k is the number of clusters. This indicates an exponential increase in the running time with the n and p.

To find the optimal number of clusters for the full dimensional and PCA-reduced data, the elbow method, silhouette score plot, grid search, and the clusteval library were tested. In the elbow method, the average distance between data points and the centroids of their clusters is plotted. The optimal value is said to be the point where this distance or inertia levels off or the point the farthest from a hypothetical line drawn from the first to the last points. However, these plots did not showcase a distinct elbow point due to the fluctuations in the inertia. The plot of the silhouette score for the number of clusters in the range 3-20 could be utilized for the same purpose where the k value corresponding to the highest silhouette score is selected.

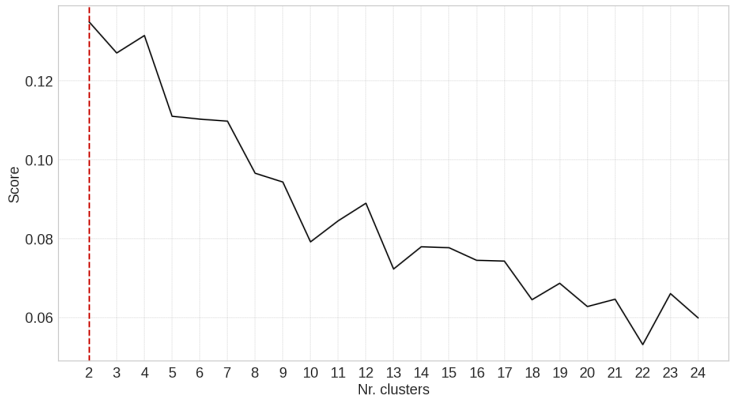

Figure 14: Silhouette score graph output of clusteval

The clusteval library uses the same logic to directly output the optimal number of clusters along with an explanatory silhouette score plot (Figure 14). Lastly, grid search has been attempted to find the optimal combination of hyperparameters (the deciding values for the algorithm) for the k-means clustering on the two sets of data. A 'grid' of the hyperparameter values is made and all possible combinations of those are 'searched' for to run to find the best set of hyperparameters. These outputs were validated with silhouette scores to decide which one resulted in the highest scores. The results of the silhouette

score (refer to Section 4 for more details on the metrics) are given in Table 1. The score was visualized in Figure 27 with the yellow-brick library.

K-means clustering with 5 clusters was run on the PCA-reduced dataset. The cluster assignments changed but not drastically. The percentage of matching cluster assignments between the pairs of observations was 89.38% that is 188301 out of 210681 (459*459). This was calculated by making two matrices for each of the two runs by populating it with 1 if elements i,j (indices) were in the same cluster else with 0. The clusters contained were nearly distributed however with the original dataset, solely one data point (396th) was a part of the fifth cluster which could indicate it is a potential outlier aligning with the results in Section 4.

Therefore, it can be said that PCA captured a substantial portion of the original variability in the data due to the similarity in the clustering obtained, however, inevitably, some structure and information have been lost due to the slight differences. The first three components or principal components seem to jointly retain a substantial portion of the original variance according to this test. A slight compromise as such can be made for the massive reduction in the number of features. The experiment was performed again with the optimal value of clusters-3 with both PCA-reduced and t-SNE-reduced data (Figure 15) against the original data (Figure 28). For the t-SNE data, only the silhouette score plot was used. The percentages of matching clusters were observed to be 96.9% (204145) and 87.57% (184499) respectively.

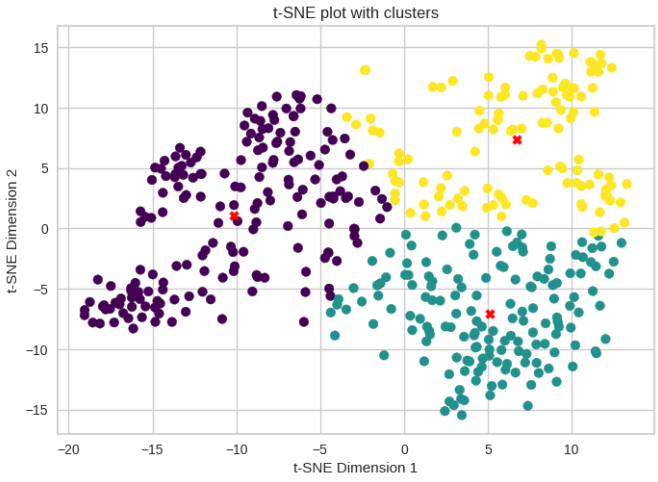

Figure 15: K-means on t-SNE reduced data with k=3

The initial cluster assignments impact the results of the clustering algorithm. To verify this, the random seed for initialization was changed 10 times to a randomly generated number in the range- 1-50, to perform clustering on the original data 10 times. The matrices described before to understand the cluster assignments of elements were made each time and stored a matrix altogether. The percentage of pairs of observations in the same cluster in all 10 runs out of the pairs in the same cluster in at least one run is 90.81%. It means that 90.81% of pairs remained in the initial cluster irrespective of the centroid initialization proving the robustness and reliability of the algorithm to the values of the initial cluster centers. It was repeated with the PCA-reduced data and an extremely high 97.66% output was observed. This indicates consistency and stability in the clustering results. Although the high consistency in the results across different initializations implies that k-means is converging to a relatively stable solution, we can not ensure the accuracy of the resulting clusters.

3.4 Single Linkage Clustering

Single linkage clustering has been performed using the functions from the Scipy library. To ensure consistent comparison between the algorithms, the value of k was set to 3. To the linkage function, the PCA-reduced data was passed along with the method that was set as 'single' for single linkage clustering to be performed.

The result of this function is a matrix of dimensions 458 (n-1) x 4. The matrix is populated as follows- each row represents the merged clusters in an iteration, the first two columns represent the indices of these two clusters, the third column represents the distance between the two clusters, and the last column represents the number of observations in the new cluster [13]. The time complexity associated is $O(n^2)$. This matrix is used for assigning cluster labels to the observations using the 'maxclust' criterion. This criterion refers to that of determining the maximum number of clusters during cluster assignment which in this case, would be to use the k (3) value given.

The number of matching cluster assignments with those from the clustering on the PCA-reduced data was compared and was found to be 77755 which is 36.91%. As only 3 clusters (0, 1, 2) are formed in this technique that too with just an observation each in 2 of the clusters, the other observations in the bigger cluster managed to match with k-means output to a third approximately. The two clusterings were visualized as a scatterplot and are given in Figures 16 and 17. Silhouette score was calculated to assess the single linkage clustering performed. The optimal value of k for the same was verified using the silhouette

score plot to be 3 as well (Figure 29).

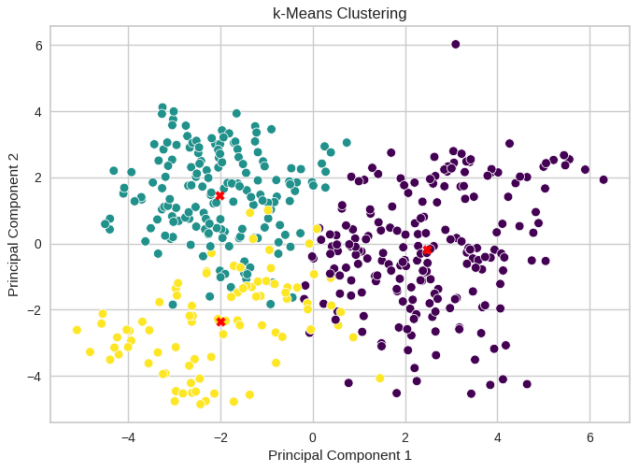

Figure 16: Scatterplot for k-means with PCA-reduced data

This process was repeated on the original dataset and on the dataset that was reduced by t-SNE. However, for these two clusterings, the optimal number of clusters was found using a silhouette score plot (Figures 30 and 31) and then run. A tree-like visual representation of the clustering in hierarchical algorithms is given in a dendrogram. It is given in Figure 32 but looks messy due to the many observations. It showcases the merging of the clusters and the distances between them.

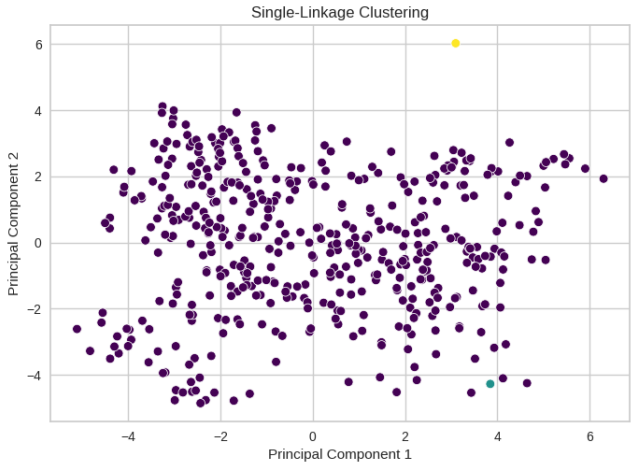

Figure 17: 2D scatterplot for single-linkage with PCA-reduced data

The clusters produced by K-means clustering tend to be convex and spherical since it aims to minimize the within-cluster sum of squares owing to the Euclidean distance used and the uniform cluster sizes. In contrast, single linkage clustering can produce chain-like or elongated clusters due to its linkage criterion. It is robust to initialization and can handle clusters of small sizes and non-convex shapes. K-means compares Euclidean distances between data points and cluster centroids, while single linkage clustering compares the same between the nearest data points in different clusters.

3.5 Gaussian Mixture Model and Expectation-Maximization Algorithm

The Gaussian mixture model from scikit-learn was instantiated on the three datasets created using the expectation-maximization algorithm. The number of components was set for the models and the initialization algorithm for the means and covariances was chosen to be k-means++ as before as the default k-means is slower. 'k' number of multivariate Gaussian distributions was initialized as the priors. The bell shape curve of the data was made and given in Figure 18.

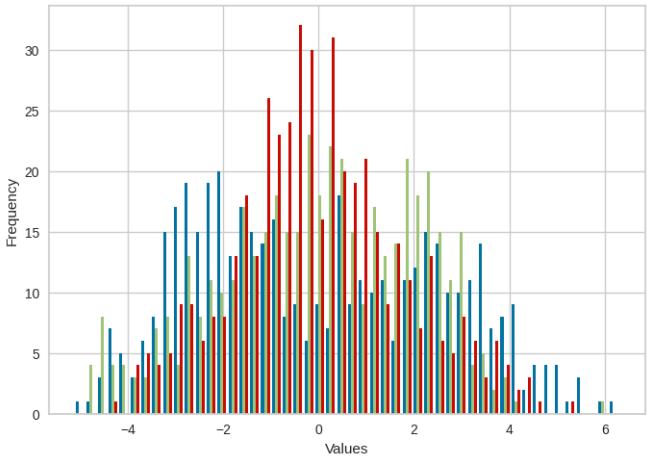

Figure 18: Histogram of the Data

When the model is fit on a dataset, the means, covariances, and mixing proportions that maximize the likelihood of the data given the model are found. The steps of expectation and maximization are run iteratively until the maximum number of iterations is reached or there is no significant increase in the likelihood of the data. Confidence ellipses were built around clusters based

on the values of the mean of the clusters and the covariance matrix for the shape and orientation. The length and orientation of the major and minor axes of the ellipse can be derived from the eigenvalues and eigenvectors of the covariance matrix. Larger eigenvalues indicate greater variability along the corresponding eigenvectors and longer axes. The centres of the clusters are the means associated with each of those Gaussian components.

The probabilistic assignments of the observations to the clusters are computed along with the labels of the clusters each point is associated with. For the case with PCA-reduced data and 3 clusters, a loop that checks the probabilities associated with the cluster assignments was written to determine the percentage of observations that are assigned to two or more clusters with a probability greater than 1%. The value was outputted as 41.18%. The assignments are useful in assessing the robustness of the algorithm and the trends in the cluster assignment. The complexity of the data reflected in the ambiguity of the cluster assignments is brought forward. It is better suited than k-means for non-circular and arbitrary data along with overlapping clusters. K-means doesn't account for the variance, unlike GMM.

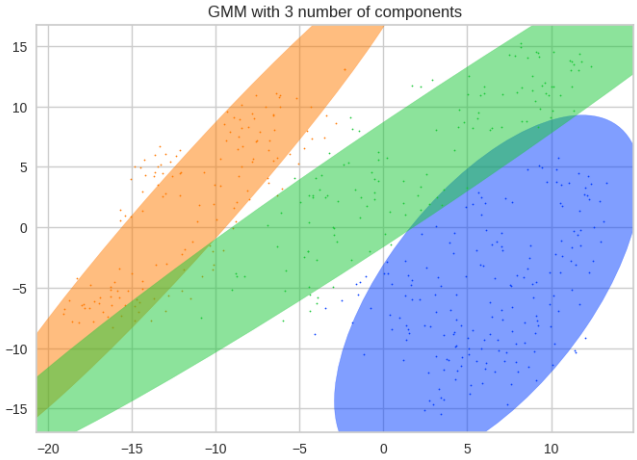

Figure 19: GMM output for t-SNE reduced data

A grid search was performed to find the ideal number of components or clusters to form from a graph of the number of clusters and the BIC score. BIC or Bayes Information Criterion refers to the scoring criteria for models where models with lower complexity and good fit are given a lower score. The lower the BIC, the better the model [22]. BIC is given by the formula-
$BIC = -2 * log - likelihood + d * \log(n)$ [23] where n is the number of

observations and d is the total number of parameters. In a GMM, d = K * (D * (D+3)/2) where D = the number of dimensions/features and K is the number of components/clusters. More parameters are penalized by the term- d * log(n) to avoid overfitting. The results on the t-SNE reduced data and the clusters are given in Figures 19 and 20. The other clusterings are given in Figures 33 and 34. The optimal 'k' value was found to be 3 for all the datasets, with which the algorithm was run for those datasets again.

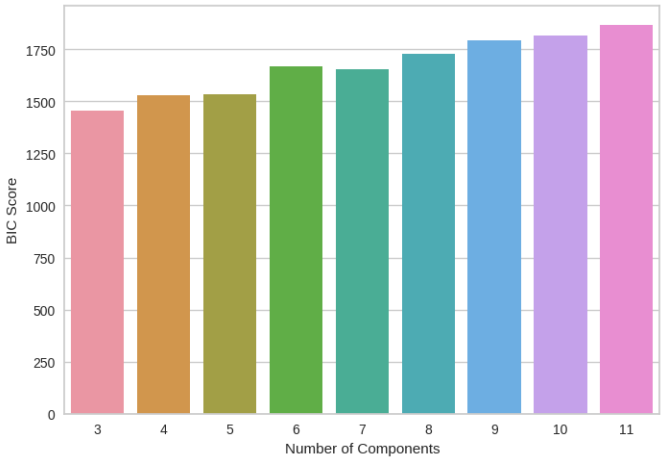

Figure 20: BIC score for t-SNE reduced data

3.6 DBSCAN

DBSCAN aims to find samples in high-density regions to cluster them so, it is suitable for data where similar data points are of similar density. DBSCAN models were instantiated from the scikit-learn library. The eps and min_samples parameters were configured after searching over a grid of parameter values but, without grid search. The silhouette score associated with each of the combinations of the parameters is calculated and the values associated with the highest silhouette score are considered the optimal values. The parameter grid was made with commonly used values for this algorithm. The other parameters were left at their default values.

The range for the value of epsilon was chosen by plotting a graph of the number of neighbours to consider and the distances to the second nearest neighbour from a K-Nearest-Neighbours (KNN) classifier. KNN is a supervised technique and it finds k number of nearest neighbours for each of the data points [24]. The value of k-5 was considered keeping in mind the size of the dataset. The value of

10 was tried but it gave the same output. The range is taken to be the values of a steep increase in the distance. A large value of epsilon/eps but a low value of the minimum number of samples gives rise to the worst-case space complexity is $O(n^2)$. However, the minimum number of points should increase with the size of the dataset A DBSCAN model is fit with these optimal values on each of the three datasets created. The output

For a random data point selected, its neighbouring points are found considering the values of eps and the value of minimum neighbouring points. If these conditions are met, the point becomes the first point in the first cluster and marked a core point else, it is marked as noise. Points marked as noise are marked to be in cluster -1 in the scikit-learn implementation. Further points are iteratively added to clusters around them until all data points within the eps distance are considered. Subsequently, another un-visited data point and its neighbouring data points are formed into a cluster if it is not marked as noise. This process continues until all data points are visited [10]. The model was fitted with the three datasets with the parameters associated with the highest silhouette score.

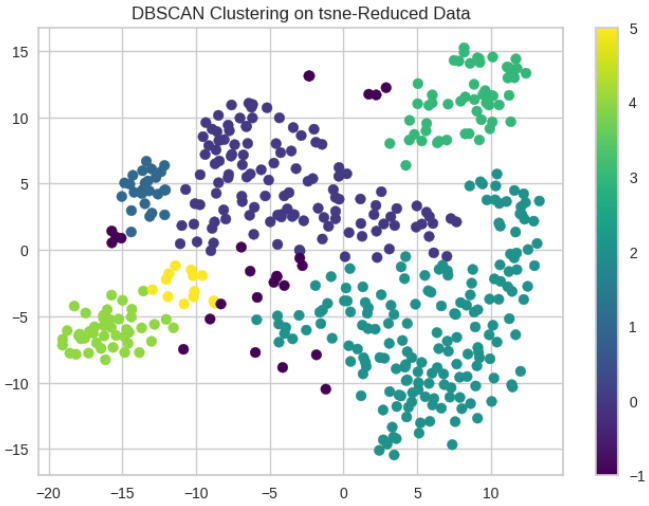

Figure 21: DBSCAN for t-SNE reduced data

The optimal parameters were: original data- {'eps': 10, 'min_samples': 3}, PCA-reduced data- {'eps': 1.5000000000000002, 'min_samples': 3}, and t-SNE reduced data- {'eps': 1.9249999999999978, 'min_samples': 8}. For the first two cases, two clusters were formed including one which classified one data point as an outlier (labeled -1). In the t-SNE case, 6 clusters were formed- 5 for the data

points and one marking the outliers. The output visualization for t-SNE reduced data is given in Figure 21. The visualizations for the others are given in Figures 36 and 37 while Figure 35 shows the KNN output for the original dataset.

3.7 BIRCH

BIRCH is said to be suitable for large datasets so it was implemented for the datasets created using its scikit-learn implementation. The cluster centres are extracted from the clustering feature tree that is created. The values of the parameters are searched from a grid of commonly used suitable values by selecting those that produce the highest silhouette score (or good fit). Parameters tweaked are- threshold- maximum radius of subclusters within a cluster, branching factor- maximum number of subclusters in each node which if exceeded would distribute the subclusters and split the node into two, and the number of clusters.

The CF Tree described in Section 2.1.6 is built by recursively creating subclusters by iterating over the data (Figure 38). Appropriate leaf nodes are found for each data point based on the Euclidean distance between them while ensuring it is less than the threshold. Linear sum, squared sum, and the number of data points in that subcluster are noted and updated upon insertion [17, 25]. Compact clusters representing the dense regions of the data are captured from the subclusters, reducing the dataset into concentrated clusters. This is referred to as global clustering. The model has been fitted to the three datasets with the optimal values found for each associated with the highest silhouette score.

The best set of parameters are as follows- for the original data- {'branching_factor': 3, 'n_clusters': 2, 'threshold': 0.01}, for PCA-reduced data- {'branching_factor': 4, 'n_clusters': 2, 'threshold': 2.0}, and t-SNE reduced data- {'branching_factor': 4, 'n_clusters': 4, 'threshold': 1.0}. The first two cases have two clusters- 0 and 1 while the last case created 4 clusters. The output of BIRCH for the PCA-reduced dataset matches that of the single linkage clustering on it if the execution stopped at two clusters whereas for the t-SNE reduced data, the clustering is similar to that by k-means on it. The clustering visualization for the datasets is given in Figures 22, 23, and 39. All visualizations can be inferred from the Google Colab link attached.

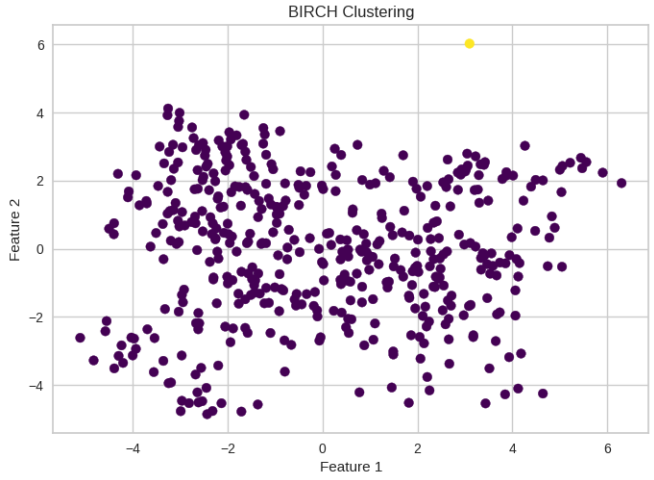

Figure 22: BIRCH on PCA-reduced data

Figure 23: BIRCH on t-SNE reduced data

4 Interpretation of the Results

After the dimension reduction algorithms, Dimensions of the original dataset:(459, 2207), dimensions of the PCA-reduced dataset: (459, 3), dimensions of the t-SNE-reduced dataset: (459, 2). The clustering algorithms implemented were tested for their performance on the full-dimensional dataset

and its two reduced versions. All algorithms were evaluated using the silhouette score. Silhouette score indicates the goodness of the clustering performed and [26] how suitable a data point is in a cluster rather than the others.

The distance of a data point from its neighbours in the same cluster (a) and the distance to other clusters (b) are computed and averaged. Silhouette Score for a data point = b-a / max(a,b) The values range from -1 to 1 where positive values indicate the rightness of the cluster assignments while a score of 0 happens when data points are equally close to two clusters. Higher scores demonstrate defined and well-separated clusters based on which, the performance of the algorithms can be compared across the datasets. Usually, a silhouette score above 0.5-0.7 [27] is considered to be corresponding to good clustering.

The k-means clustering of all the pairs of observations in the full-dimensional and reduced datasets was compared using Adjusted Random Score/Index. Its values also range from -1 to 1 that is, from disagreement, no chance of agreement, to perfectly identical. The outliers in each cluster were attempted to be found to understand the quality of the clusters or possibly view the outliers or noise in the dataset.

Method	k- full data	k- reduced data	Full Data	Reduced Data	Adjusted Rand Score
Silhouette Score Plot	4	3	0.13	0.34	0.72
Grid Search	15	15	0.07	0.29	0.45
Clusteval Library	2	3	0.13	0.35	0.66
Initial Configuration	5	5	0.12	0.32	0.70

Table 1: Silhouette score related to experiments for optimal k values

The highest silhouette score corresponds to 3 clusters for both datasets and so is the adjusted rand score. A silhouette score of 0.13 for a high-dimensional dataset- the original dataset indicates poor cluster quality and separation while a score of 0.35 for the PCA-reduced dataset suggests moderate cluster quality and separation. As a result, k = 3 was selected for both cases for consistency and the Silhouette score plot was selected for further optimal 'k' value searches. The clustering agreement was the best in the first case too, but resembles the initial configuration's output. It resembles clusteval's output too as the same algorithm is implemented behind the hood with slightly different parameters.

For the first case, 3 outliers were found in 3 of the 4 clusters for the original dataset. In the second case, kmeans++ was found ideal for both the datasets but, for the original dataset maximum of 300 iterations were found, 1 cluster

contained 2 outliers and another had 1 outlier. For the third case, out of the 2 clusters, one was marked to have 3 outliers. For the reduced dataset, only one observation was marked as an outlier in all three cases. Overall, observations 95, 97, and 369 were marked outliers in all cases with at least one of the dataset variants and in the initial and ideal cases. However, with t-SNE in the ideal case with 4 clusters, no outliers were observed.

The adjusted rand score for the ideal case between the original and clusters of the PCA-reduced dataset is 0.93 but it is 0.288 between the original and t-SNE reduced dataset. This can also be observed from the previous section where the percentage of matching clusters was better in the former case. The low ARS between the original data and t-SNE-based clustering could indicate a loss of information compared to PCA or more concentration on local instead of the global structure than required.

GMM is evaluated using BIC and Akaike information criterion (AIC) (Table 2) along with silhouette score. AIC is similar to BIC but generally used for time series or small datasets so it has not been stressed. BIC is preferred over AIC for the true model selection. The log-likelihood of a GMM measures the fit of the model to the data. Its value is said to be relative to datasets so cannot be extended to other datasets or even their reduced versions. As a reason, the values are calculated but not compared.

Data	Number of Components	AIC	BIC
Original Data	3	3418814.42	33627800.47
PCA-Reduced Data	3	5795.05	5914.8
t-SNE Reduced Data	3	6215.74	6285.93

Table 2: GMM Results

The number of components and covariance type used is the same in all three cases, however, the lowest AIC and BIC are preferred therefore, the PCA-reduced data is clustered better abstractly as values cannot be directly compared due to the dataset variations.

Below is the table (Table 3) outlining the best silhouette scores that could be obtained for each of the datasets with each clustering technique. A silhouette score plot of the ideal k-means run on the PCA-reduced dataset is given in Figure 27.

Algorithm	Original Data	PCA-Reduced Data	t-SNE Reduced Data
K-Means	0.13	0.35	0.46
Single-Linkage	0.34	0.19	0.12
GMM	0.08	0.29	0.35
DBSCAN	0.71	0.40	0.32
BIRCH	0.12	0.40	0.45

Table 3: Best Sillhoutte Scores

For K-means, GMM, and BIRCH models, the best performance was with the t-SNE reduced data. With single linkage clustering, the overall performance is underwhelming but the best performance was observed with the original full-dimensional dataset. Similarly, the best DBSCAN results were with the original dataset. On the original dataset, DBSCAN performed the best, on the PCA-reduced dataset, both DBSCAN and BIRCH had similar results, and on the t-SNE-reduced dataset, k-means clustering performed the best.
Single-linkage clustering was the worst-performing algorithm averaged across the three datasets.

K-Means achieved the highest silhouette scores on the t-SNE reduced data, then the PCA-reduced data, and lowest on the original Data. Single linkage had the highest score on the original data followed by the PCA-reduced data and t-SNE-reduced data. GMM displayed relatively lower scores and poor performances across all datasets. DBSCAN's performance decreased on the reduced datasets while BIRCH had comparatively lower silhouette scores on all three datasets. Overall, the highest silhouette score or clearest clustering on the dataset was observed with DBSCAN on the full-dimensional dataset which could be attributed to the fact that DBSCAN produces arbitrarily-shaped clusters thus recognizing better clusters from this dataset.

On average, the best performance of the algorithms was observed on the t-SNE reduced dataset, and the worst on the high-dimensional original data showing that a large number of features are indeed not preferred for clustering. Clusters in the same parts of the t-SNE data tend to belong to the same clusters compared to PCA as it is very random there but the former appears more put together and systematic. Additionally, it's preferable to consider the domain knowledge to make a well-versed decision regarding the algorithms and their results on the dataset and its reduced variants.

5 Comparison of the Algorithms

K-means clustering is intuitive and easy to implement even without any libraries, unlike the other algorithms which are cumbersome to implement from scratch. The pace of execution was comparably fair for all techniques on the reduced datasets but, fitting the GMM, BIRCH, and DBSCAN models on the original dataset was a hassle as they ran for longer times due to their dimensionality. A high number of features slowed down the model and added ambiguity however, performance improved for reduced datasets. The clusters produced are approximately the same size and spherical so overlapping clusters were found. In single Linkage clustering, smaller clusters could potentially unravel similarities between data points and the hierarchical structure is intuitive in most cases. For oblong clusters, k-means would be inaccurate, and this problem is resolved by using GMM or DBSCAN.

DBSCAN performs better on arbitrarily shaped data and can identify outliers. It does not require a predefined number of clusters like k-means clustering or GMM. Another plus is the recognition of outliers as long as a spatial density difference is recognized. Although the given dataset is comprised of numerical values, it is to be noted that BIRCH does not work on categorical variables directly. BIRCH is a memory-efficient technique [15] due to the summaries it creates and is better for large datasets as seen with our data. The number of clusters needs to be mentioned with k-means which is not required for the other algorithms however, it is good practice to use parameters search techniques for an optimized run.

Single linkage clustering and GMM are slower for larger datasets while the others seem to be less computationally expensive so faster. The clusters produced by k-means are easily understandable and interpretable while the dendrogram improves the understanding of the single linkage clustering. The hierarchical structure can provide insights into cluster relationships just as with BIRCH. GMM is useful if the probabilities of observations concerning each of the clusters are important compared to the exact cluster assignments as they would aid in showing the probabilities associated with overlapping clusters.

6 Discussion

6.1 Limitations

The description of the features and other background details of the dataset are unknown due to which feature selection could not be performed. In addition, pre-processing steps were limited to checking for missing and duplicate values and the range of the features. In supervised learning algorithms, it is easier to judge and test the accuracy of the algorithm since the labels are known. However, with unsupervised algorithms like clustering, patterns found by the algorithm cannot be manually validated at all.

Clustering as a process does not exist in supervised learning because data points are classified into different classes that correspond to each of the unique values of the target attributes. They can only be evaluated using criteria such as silhouette score or AIC and BIC but they do not account for the true groups as they are unknown. Finding the optimal number of clusters is a challenge with complex high-dimensional datasets so the clustering made for the original dataset even with the optimal values found, was worse than what was found for the reduced datasets. Outliers or noise can significantly impact clustering results except in the case of DBSCAN as it is robust to outliers. Thus, outlier detection techniques would need to be implemented to avoid this problem.

Although clustering is a simple process, the algorithms each have their downsides. Algorithms are highly sensitive to their initial values thus extra care has to be taken to ensure the initialization is done after using relevant values for the dataset's size and background. K-means clustering and single linkage clustering are linear algorithms along with PCA so are unfit for non-linear data. The latter could be substituted with t-SNE. K-means tends to create clusters of almost equal sizes and spherical which if not the case with the natural clustering of the data could lead to unacceptable and inaccurate results [10]. Alongside, solutions tend to converge to local optima paying less attention to the global structure of the data.

Single Linkage clustering is suitable for smaller datasets or for the ones the researcher is aware of the background because in this case, reading the complicated dendrogram did not provide good insights into the clustering. Outliers are also classified as a part of a cluster at one point or another making it unreliable with non-processed data [10]. Due to the single linkage criterion, clusters would be elongated and this is called the chaining effect. DBSCAN expects density differences to detect clusters although it is suitable for arbitrary

shaped data. It would struggle if densities don't differ much even in the presence of highly dense regions. With a large dataset, there is a risk of producing incorrect clusters as the points could be uniformly densely populated. BIRCH is a comparatively complex technique and is highly influenced by the values of its parameters such as the branching factor.

PCA could not perform optimally as a large number of principal components were associated with explaining a good amount of the variance of the original data. t-SNE was simpler to use as the divergence produced with the default perplexity was satisfactory but for better divergence, more perplexity and resources would have been required. Although it stresses the local neighbouring points more, the value of perplexity influences the balance of local and global structure's influence. Hence, it needs to be kept in mind during configuration. It is computationally expensive and slower than PCA [28]. Additionally, the variance from PCA and divergence from t-SNE cannot be compared to see which one is better.

6.2 Future Work

The project can be extended by experimenting with more clustering algorithms to assess their performance on datasets with a higher number of features such as other hierarchical methods, fuzzy clustering, affinity propagation, or OPTICS which is a modified version of DBSCAN. To solve to problem of minimal pre-processing, research could be done to discover the source of the dataset and understand its features to incorporate domain knowledge into this project. Alternatively, the same code could be adapted for a real-world dataset with enough background detail available. Ensemble clustering is rising in popularity. It is also known as consensus clustering and it combines multiple basic clustering algorithms as a unit for a reliable, stable, and robust grouping on datasets of any scale [29]. As the amount of data grows in volume and complexity, further research to improve our understanding of data and extract meaningful information will be beneficial.

7 Conclusion

The project has conducted a thorough review of clustering algorithms and the effect of dimensionality reduction algorithms on these techniques. For unlabeled data, unsupervised learning techniques have to be utilized so clustering algorithms were implemented. In comparison, the DBSCAN technique seemed

to work best for the t-SNE-reduced dataset. Between PCA and t-SNE, the dataset reduced by t-SNE had a better average score although the criteria cannot be compared to each other. All algorithms have their pros and cons which have been highlighted as intended and they are each suitable for different types of datasets. The domain of machine learning is wider so there exist many other clustering techniques that could be implemented. Clustering and dimensionality reduction techniques are vital in the analysis and understanding of complex datasets. By understanding how to effectively group and cluster data, we can develop an effective understanding of the observations and features and gain enough insights to make data-driven decisions in various domains.

8 Appendices

8.1 Appendix A: Figures

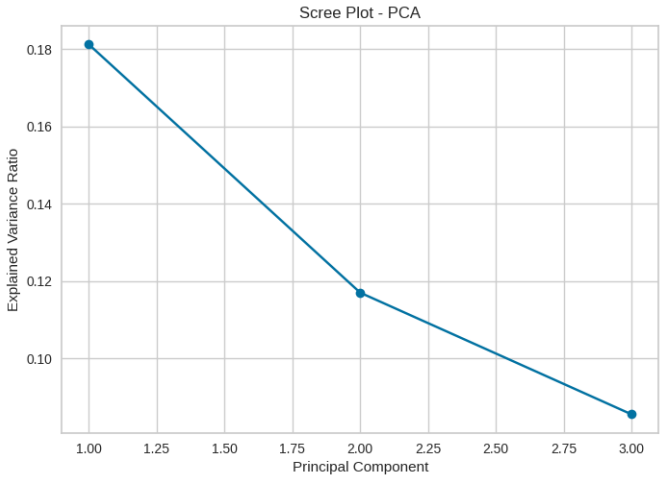

Figure 24: Principal Components and explained variance

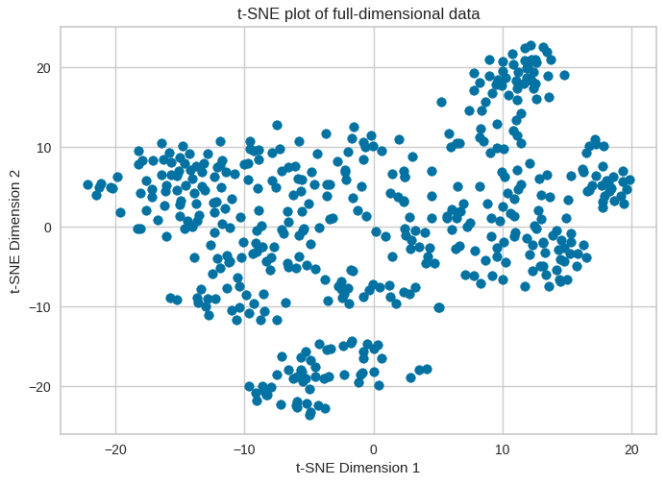

Figure 25: Inital t-SNE

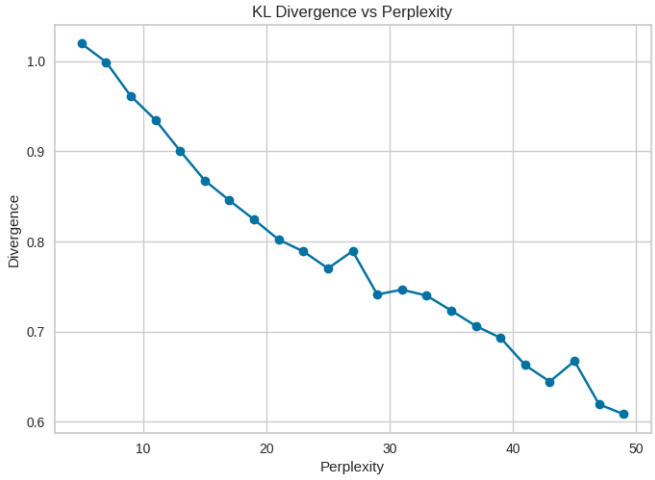

Figure 26: KL Divergence vs Perplexity

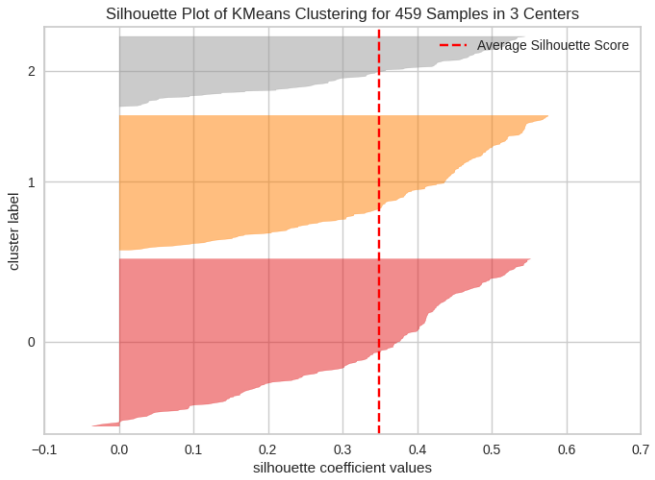

Figure 27: Silhoutte score plot for PCA-reduced data with k-means

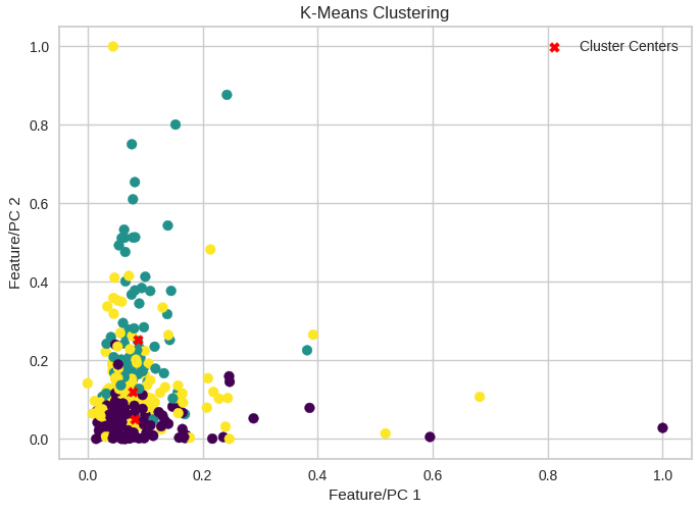

Figure 28: Final k-means on the original data

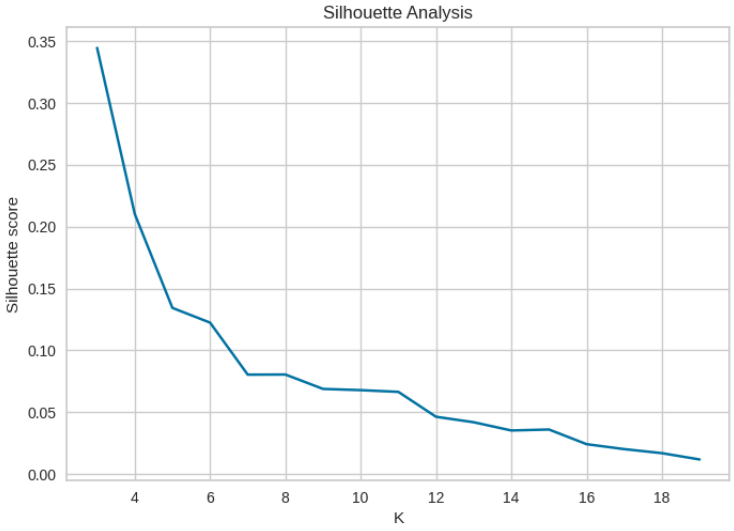

Figure 29: Silhouette Score plot with single-linkage for PCA-reduced data

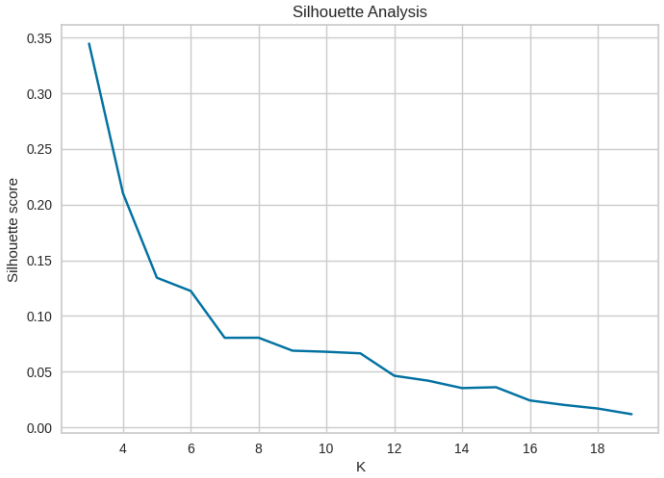

Figure 30: Silhouette Score plot with single-linkage for original data

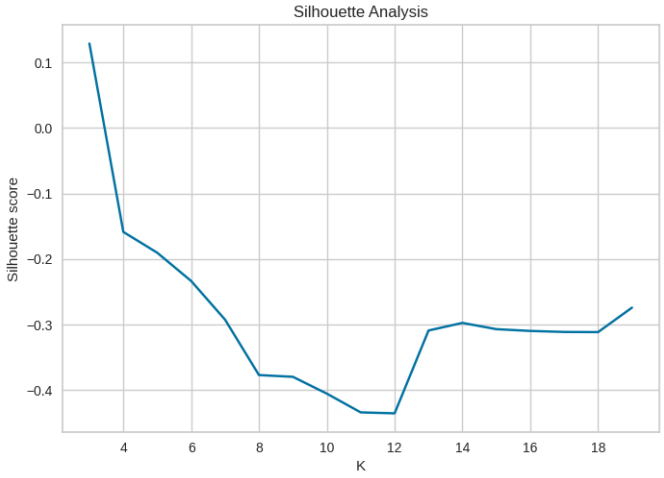

Figure 31: Silhouette Score plot with single-linkage for t-SNE-reduced data

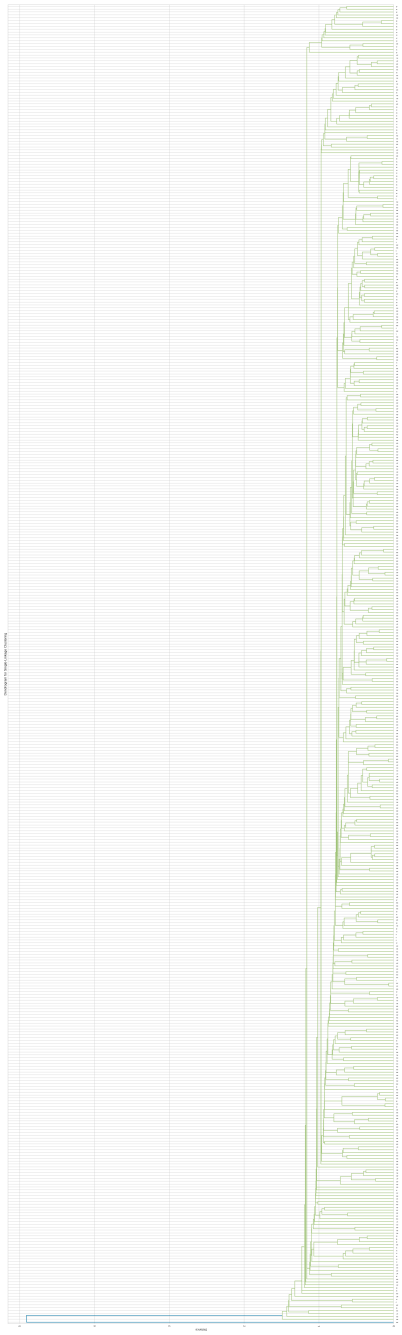

Figure 32: Dendrogram for PCA-reduced data

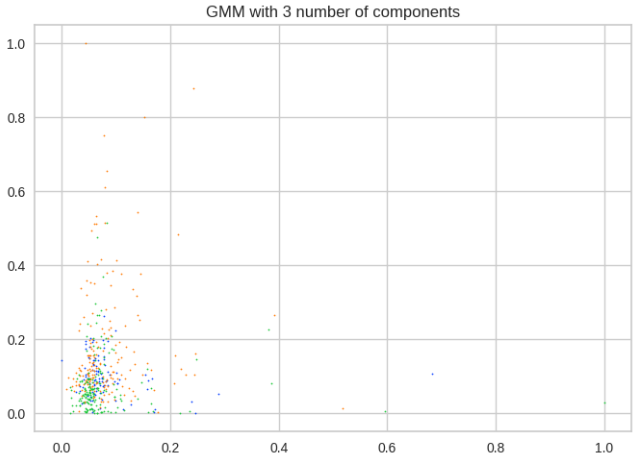

Figure 33: GMM on the original data

Figure 34: GMM on PCA-reduced data

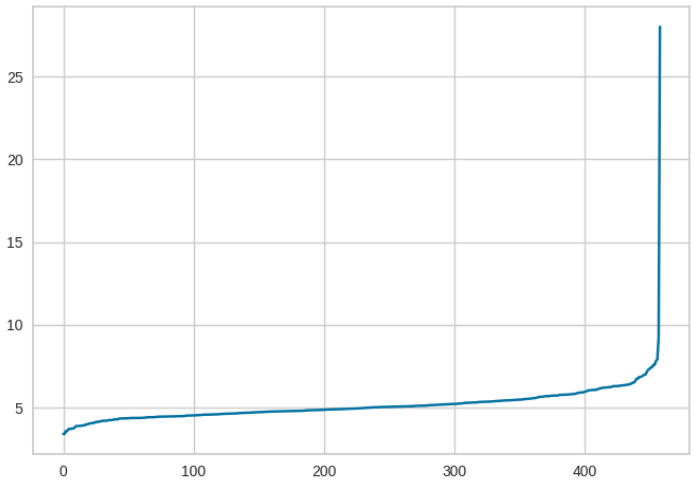

Figure 35: KNN output for the original data so range (8, 25) is chosen

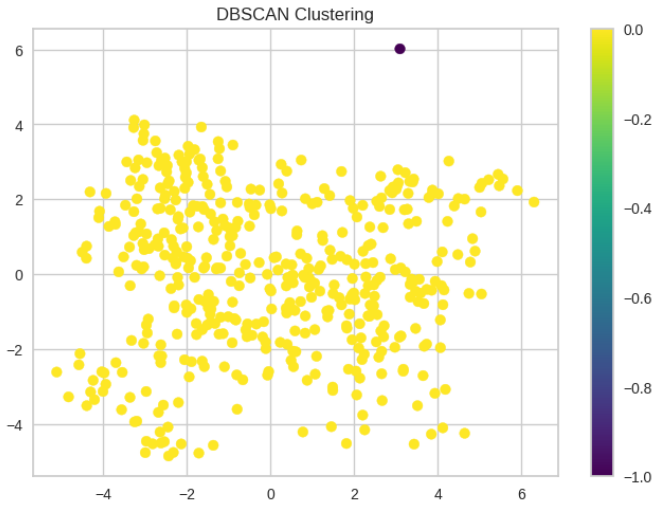

Figure 36: DBSCAN output for the original data

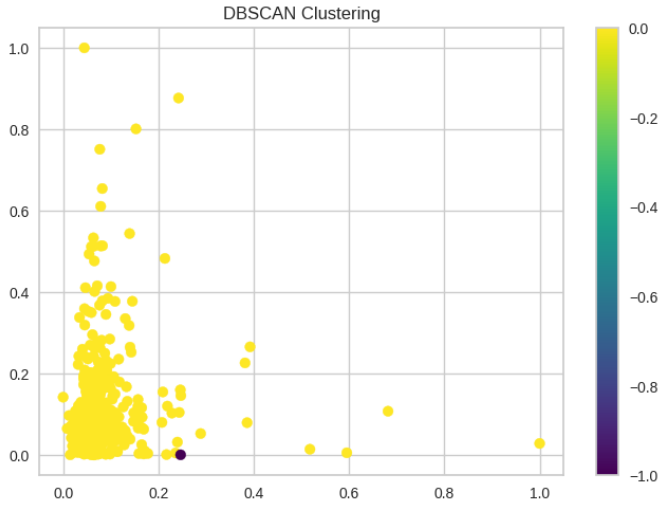

Figure 37: DBSCAN output for the PCA-reduced data

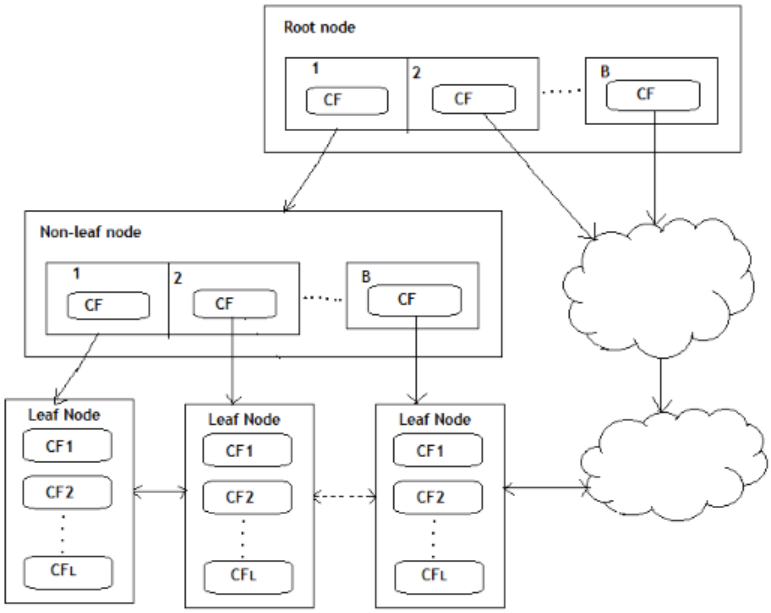

Fig: A CF tree structure

Figure 38: BIRCH- Cluster Feature Tree [30]

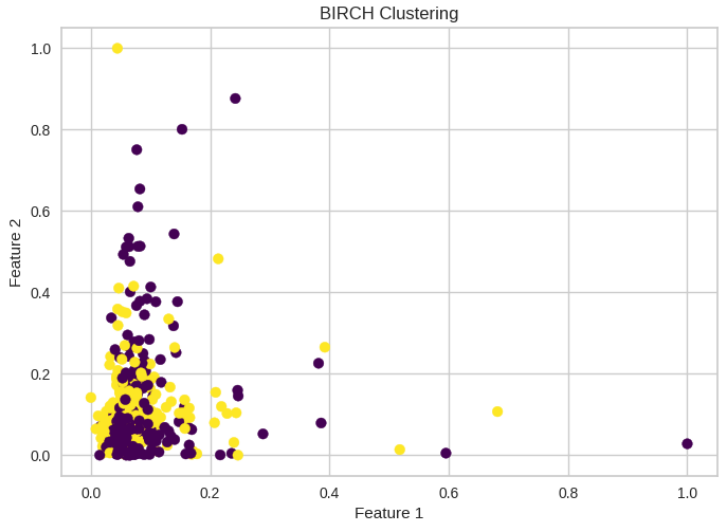

Figure 39: BIRCH output for the original data

8.2 Appendix B: Python Code

The code is available at this link. I have attached a naive Python version below but I would recommend considering viewing the above Jupyter Notebook to view the code and outputs together.

```
# -*- coding: utf-8 -*-
"""MATH572 Clustering.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1UAEFaeSWRPrzHxjYONvIWMV9KBBsna7x

# MATH572: Comparing Clustering Algorithms
"""

!pip install clusteval

# Importing necessary libraries and functions
import pandas as pd
import numpy as np
import copy
import random
from sklearn.cluster import KMeans, DBSCAN, Birch
from sklearn.decomposition import PCA
```

```
from yellowbrick.cluster import SilhouetteVisualizer
from sklearn.neighbors import LocalOutlierFactor,
    NearestNeighbors
from clusteval import clusteval
from sklearn.metrics import silhouette_score,
    adjusted_rand_score
from scipy.cluster.hierarchy import linkage, fcluster,
    dendrogram
from sklearn.mixture import GaussianMixture
from sklearn.manifold import TSNE
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import ParameterGrid
import requests
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.patches import Ellipse
from scipy import linalg

"""## Data Loading"""

data =
    pd.read_csv("https://russellmilne.com/teaching/gvrd-rangestd.csv",
    header=None) # loading the data into a dataframe
print(data.head()) # glimpse of the data

data = data.T # transposing as observations are in rows and
    features in rows
print("Dimensions␣of␣the␣dataset:␣", data.shape)
print('*'*100)
print(data.describe()) # glimpse of the data
print('*'*100)

print("Columns/Features:␣",data.columns)
print("\nDataframe␣information:")
data.info()

print("Null␣values:␣", data.isna().sum().sum()) # checling
    for null values
print("Number␣of␣duplicates:␣", data.duplicated().sum()) #
    checking for duplicates

# to check need for standardization as uptil now, there does
    not seem to be any need
# if needed, the below commented code could be run
# scaled_data = StandardScaler().fit_transform(data) # to
    standardize the features
```

```
# data = pd.DataFrame(scaled_data, columns=data.columns) #
    converting the scaled data into a dataframe from an array

print("All minimums are less than 1: ",(data.min()<1).all())
    # minimum is 0 and all minimums are checked to be less
    than 1
print("All maximums are less than or equal to 1:
    ",(data.max()<=1).all()) # all maximums are checked to be
    less than or equal to 1
# Both are True so no scaling is needed

plt.scatter(data.iloc[:, 0], data.iloc[:, 1]) # for naively
    visualizing the entire data in 2 dimensional space
plt.title('2D Scatrerplot of Data')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()

"""## Principal Component Analysis"""

pca = PCA(n_components=3) # PCA object instantiated with 3
    components as in project
reduced_data = pca.fit_transform(data) # principal component
    analysis on the data; fit model and reduce dimensions

# scree plot
plt.plot(np.arange(1, len(pca.explained_variance_ratio_) +
    1), pca.explained_variance_ratio_, 'bo-')
plt.xlabel('Principal Component')
plt.ylabel('Explained Variance Ratio')
plt.title('Scree Plot - PCA')
plt.show()

# PCA is instantiated again without any configuration to see
    how many principal compoents would be ideal
pca1 = PCA()
pca1.fit_transform(data)
explained_variance_ratio = pca1.explained_variance_ratio_ #
    proportion of the total variance in the data explained by
    first 50 principal components
cum_var = np.cumsum(explained_variance_ratio) # cumulative
    (summed) explained variance
# represents the number of principal components surpassing
    the threshold
num_components = np.sum(cum_var >= 0.9)
```

```
# scree plot to see how many PCs to select, not much
    difference is found
plt.plot(np.arange(1, len(explained_variance_ratio) + 1),
    explained_variance_ratio, 'b-')
plt.plot(np.arange(len(explained_variance_ratio)), cum_var,
    'r-')
plt.axhline(0.9, label='90% threshold', color='black') # to
    mark the 90% threshold
plt.xlabel('Principal Component')
plt.ylabel('Explained Variance Ratio')
plt.title('Scree Plot - PCA')
plt.show()

reduced_data = pd.DataFrame(reduced_data) # converting to a
    dataframe

# Scatter plot with the reduced data
plt.scatter(reduced_data.iloc[:, 0], reduced_data.iloc[:,
    1]) # for the data
plt.title(f'PCA')
plt.xlabel('PC 1')
plt.ylabel('PC 2')
plt.show()

"""## Using t-SNE (t-distributed Stochastic Neighbor
    Embedding)"""

# t-sne to find clusters in high-dimensional datasets
tsne_obj = TSNE(n_components=2, n_iter=2000, random_state=0)
    # n_components parameter=2 for a 2D representation,
    random_state for reproducibility of the run
# n_iter is the number of iterations to run the algorithm for
tsne = tsne_obj.fit_transform(data) # applying t-sne on the
    original data for a reduced-dimensional dataset
tsne = pd.DataFrame(tsne)

# Plotting the t-sne results
plt.scatter(tsne.iloc[:, 0], tsne.iloc[:, 1]) # both the
    tsne dimensions
plt.title('t-SNE plot of full-dimensional data')
plt.xlabel('t-SNE Dimension 1')
plt.ylabel('t-SNE Dimension 2')
plt.show()

print("\n KL divergence score: ", tsne_obj.kl_divergence_)

perplexity = np.arange(5, 50, 2) # range of perplexities
```

```
divergence = [] # to store KL divergence values
for i in perplexity:
    model = TSNE(n_components=2, init="pca", perplexity=i) #
       T-SNE model is fit with type pca to initially start
       with pca type reduction
    reduced = model.fit_transform(data)
    divergence.append(model.kl_divergence_)

plt.plot(perplexity, divergence, 'bo-')
plt.title("KL Divergence vs Perplexity")
plt.xlabel("Perplexity")
plt.ylabel("Divergence")

# t-sne to find clusters in high-dimensional datasets
tsne_obj = TSNE(n_components=2, n_iter=2000, random_state=0,
   perplexity=40) # n_components parameter=2 for a 2D
   representation, random_state for reproducibility of the
   run
# n_iter is the number of iterations to run the algorithm for
tsne = tsne_obj.fit_transform(data) # applying t-sne on the
   original data for a reduced-dimensional dataset
tsne = pd.DataFrame(tsne)

# t-sne to find clusters in high-dimensional datasets
tsne_obj = TSNE(n_components=2, n_iter=2000, random_state=0,
   perplexity=40) # n_components parameter=2 for a 2D
   representation, random_state for reproducibility of the
   run
# n_iter is the number of iterations to run the algorithm for
tsne = tsne_obj.fit_transform(data) # applying t-sne on the
   original data for a reduced-dimensional dataset
tsne = pd.DataFrame(tsne)

# Plotting the t-sne results
plt.scatter(tsne.iloc[:, 0], tsne.iloc[:, 1]) # both the
   tsne dimensions
plt.title('t-SNE plot of full-dimensional data')
plt.xlabel('t-SNE Dimension 1')
plt.ylabel('t-SNE Dimension 2')
plt.show()

print("KL divergence score: ", tsne_obj.kl_divergence_)

print("Dimensions of original dataset: ",data.shape)
print("\nDimensions of PCA-reduced dataset:
   ",reduced_data.shape)
print("\nDimensions of t-SNE-reduced dataset: ",tsne.shape)
```

```
"""## Comparing K-Means Clustering on Original and
   Reduced-dimensional Dataset
Reduced-dimensional dataset has been obtained from principal
   component analysis

### Clustering with randomly-initialized k value
"""

# Running k-means clusering on the dataset with a randomly
   assigned initial number of clusters
k1 = 5 # number of clusters
kmeans1 = KMeans(n_clusters=k1, random_state=0, n_init=10)
kmeans1.fit(data) # fitting a k-means model; running the
   algorithm on this data.
clusters_original = kmeans1.labels_ # clusters assgined to
   the data points
print("Uniue␣cluster␣labels:␣",np.unique(clusters_original))
print("\nCluster␣
   counts:\n",pd.Series(clusters_original).value_counts())

n = data.shape[0] # number of rows/observations
M = np.zeros((n, n)) # M matrix with all 0s

# to populate M with 1 if observations at i and j are in the
   same cluster and 0 is left if they are in different
   clusters
for i in range(n):
    for j in range(n):
        if clusters_original[i] == clusters_original[j]:
            M[i][j] = 1

k2 = 5 # number of clusters
kmeans2 = KMeans(n_clusters=k2, random_state=0, n_init=10)
clusters_reduced = kmeans2.fit_predict(reduced_data) # the
   new clusters lables assigned
print("Uniue␣cluster␣labels:␣",np.unique(clusters_original))
print("\nNew␣cluster␣
   counts:\n",pd.Series(clusters_reduced).value_counts())

P = np.zeros((n, n))
# to populate P with 1 if observations at i and j are in the
   same cluster and 0 is left if they are in different
   clusters
for i in range(n):
    for j in range(n):
        if clusters_reduced[i] == clusters_reduced[j]:
```

```
            P[i][j] = 1

# checking the number of matching cluster assignments from M
    and P
matches = np.sum(M == P)
print("Number of matching cluster assignments between M and
    P:", matches)
print("Percentage of matching cluster assignments = ",
    round(matches/(n*n) * 100,2))

def testingfunc (data1, clusters, cluster_centers, k1):
  """
    Testing the clustering algorithms' outputs on a dataset
  """
  silhouette = silhouette_score(data1, clusters) # silhoutte
     score of the clustering on the dataset
  print("Silhouette coefficient:", silhouette)

  outlier_score = LocalOutlierFactor().fit_predict(data1) #
     outlier detection in the clusters
  # to find the number of outliers in each of the clusters
  for cluster in range(k1):
    cluster_indices = np.where(clusters_original ==
       cluster)[0]
    cluster_outlier_score = outlier_score[cluster_indices]
    outliers = cluster_indices[cluster_outlier_score == -1]
    print("outliers in cluster", cluster, ":", outliers)

  # Scatter plot with data by only the first 2
     features/dimensions or principal components and the
     cluster centers
  plt.scatter(data1.iloc[:, 0], data1.iloc[:, 1],
     c=clusters, cmap='viridis') # for the data
  plt.scatter(cluster_centers[:, 0], cluster_centers[:, 1],
     c='red', marker='X', label='Cluster Centers') # for the
     cluster centres
  plt.title(f'K-Means Clustering')
  plt.xlabel('Feature/PC 1')
  plt.ylabel('Feature/PC 2')
  plt.legend()
  plt.show()

testingfunc(data, clusters_original,
    kmeans1.cluster_centers_, k1)

testingfunc(reduced_data, clusters_reduced,
    kmeans2.cluster_centers_, k2)
```

```
ars = adjusted_rand_score(clusters_original,
    clusters_reduced) # comparing the two clusterings
print("Adjusted␣Rand␣Score:", ars)

"""### Finding Optimal The Number of Clusters

#### Elbow Method and Silhoutte Score Plot
"""

# Elbow Method and Silhoutte Score Plot for the
    full-dimensiona;/original dataset
# to find the optimal number of clusters (k), the elbow
    method is used and the silhoutte score for each is
    compared
silhouette = [] # for the silhoutte score
inertia = [] # list of squares um of the distances of a
    point from the centroid
k = range(3,20) # clusters values from 4-20 are checked
# for each number of cluster, k-means is run and and inertia
    is calculated
# low inertia is preferred
for ks in k :
  kmeans = KMeans(n_clusters=ks, n_init=10, random_state=0)
  kmeans.fit(data)
  inertia.append(kmeans.inertia_)
  silhouette.append(silhouette_score(data, kmeans.labels_))

# to check for the point where the inertia is low and the
    number of clusters are balanced (elbow point)
plt.plot(k,inertia)
plt.xlabel('K')
plt.ylabel('Inertia')
plt.title('Elbow␣Method')
plt.grid(True)
plt.show()

# to check for k with highest silhoutte score
plt.plot(k, silhouette)
plt.xlabel('K')
plt.ylabel('Silhouette␣score')
plt.title('Silhouette␣Analysis')
plt.grid(True)
plt.show()

# Elbow Method and Silhoutte Score Plot for the reduced
    dataset
```

```
# to find the optimal number of clusters (k), the elbow
    method is used and the silhoutte score for each is
    compared
silhouette = [] # for the silhoutte score
inertia = [] # list of squares um of the distances of a
    point from the centroid
k = range(3,20) # clusters values from 4-20 are checked
# for each number of cluster, k-means is run and and inertia
    is calculated
# low inertia is preferred
for ks in k :
  kmeans = KMeans(n_clusters=ks, n_init=10, random_state=0)
  kmeans.fit(reduced_data)
  inertia.append(kmeans.inertia_)
  silhouette.append(silhouette_score(reduced_data,
      kmeans.labels_))

# to check for the point where the inertia is low and the
    number of clusters are balanced (elbow point)
plt.plot(k,inertia)
plt.xlabel('K')
plt.ylabel('Inertia')
plt.title('Elbow Method')
plt.grid(True)
plt.show()

# to check for k with highest silhoutte score
plt.plot(k, silhouette)
plt.xlabel('K')
plt.ylabel('Silhouette score')
plt.title('Silhouette Analysis')
plt.grid(True)
plt.show()

# optimal k value from both the analysis is decided to be 3.

# Running k-means clusering on the dataset with a randomly
    assigned initial number of clusters
k1 = 4 # number of clusters
kmeans1 = KMeans(n_clusters=k1, random_state=0, n_init=10)
kmeans1.fit(data) # fitting a k-means model; running the
    algorithm on this data.
clusters_original = kmeans1.labels_ # clusters assgined to
    the data points

k2 = 3 # number of clusters
kmeans2 = KMeans(n_clusters=k2, random_state=0, n_init=10)
```

```
clusters_reduced = kmeans2.fit_predict(reduced_data) # the
    new clusters lables assigned

testingfunc(data, clusters_original,
    kmeans1.cluster_centers_, k1)
testingfunc(reduced_data, clusters_reduced,
    kmeans2.cluster_centers_, k2)
ars = adjusted_rand_score(clusters_original,
    clusters_reduced) # comparing the two clusterings
print("Adjusted␣Rand␣Score:", ars)

"""#### Grid Search"""

# parameter grid/dictionary to test for the K-means model
param_grid = {
    'n_clusters': [4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15],
    'init': ['k-means++', 'random'],
    'max_iter': [100, 300, 500],
}

kmeans = KMeans(n_init=10) # k-means model
grid_search_km = GridSearchCV(estimator=kmeans,
    param_grid=param_grid) # to search for the optimal
    parameter values for k-means
grid_search_km.fit(data) # fitting the model on the original
    data
print("Best␣parameters␣for␣K-means:",
    grid_search_km.best_params_)

grid_search_km.fit(reduced_data) # fitting the model on the
    reduced data
print("Best␣parameters␣for␣K-means:",
    grid_search_km.best_params_)

# Running k-means clusering on the dataset with a randomly
    assigned initial number of clusters
k1 = 15 # number of clusters
kmeans1 = KMeans(init= 'k-means++', max_iter= 300,
    n_clusters=k1, random_state=0, n_init=10)
kmeans1.fit(data) # fitting a k-means model; running the
    algorithm on this data.
clusters_original = kmeans1.labels_ # clusters assgined to
    the data points

k2 = 15 # number of clusters
kmeans2 = KMeans(init= 'k-means++', max_iter= 500,
    n_clusters=k2, random_state=0, n_init=10)
```

```
clusters_reduced = kmeans2.fit_predict(reduced_data) # the
    new clusters lables assigned

testingfunc(data, clusters_original,
    kmeans1.cluster_centers_, k1)
testingfunc(reduced_data, clusters_reduced,
    kmeans2.cluster_centers_, k2)
ars = adjusted_rand_score(clusters_original,
    clusters_reduced) # comparing the two clusterings
print("Adjusted␣Rand␣Score:", ars)

"""#### Using clusteval library"""

ceval = clusteval('kmeans') # to evaluate kmeans model
results= ceval.fit(data) # to find optimal k for the
    original data
ceval.plot() # plot of silhoutte score with k value
    highlighted

results= ceval.fit(reduced_data) # to find optimal k for the
    original data
ceval.plot() # plot of silhoutte score with k value
    highlighted

# Running k-means clusering on the dataset with a randomly
    assigned initial number of clusters
k1 = 2 # number of clusters
kmeans1 = KMeans(n_clusters=k1, random_state=0, n_init=1)
kmeans1.fit(data) # fitting a k-means model; running the
    algorithm on this data.
clusters_original = kmeans1.labels_ # clusters assgined to
    the data points

k2 = 3 # number of clusters
kmeans2 = KMeans(n_clusters=k2, random_state=0, n_init=1)
clusters_reduced = kmeans2.fit_predict(reduced_data) # the
    new clusters lables assigned

testingfunc(data, clusters_original,
    kmeans1.cluster_centers_, k1)
testingfunc(reduced_data, clusters_reduced,
    kmeans2.cluster_centers_, k2)
ars = adjusted_rand_score(clusters_original,
    clusters_reduced) # comparing the two clusterings
print("Adjusted␣Rand␣Score:", ars)

"""### Ideal Run"""
```

```
# Running k-means clusering on the dataset with a randomly
    assigned initial number of clusters
k1 = 3 # number of clusters
kmeans1 = KMeans(n_clusters=k1, random_state=0, n_init=10)
kmeans1.fit(data) # fitting a k-means model; running the
    algorithm on this data.
clusters_original = kmeans1.labels_ # clusters assgined to
    the data points
print("Unique clusters:\n",np.unique(clusters_original))
print("Cluster
    counts:\n",pd.Series(clusters_original).value_counts())

n = data.shape[0] # number of rows/observations
M = np.zeros((n, n)) # M matrix with all 0s

# to populate M with 1 if observations at i and j are in the
    same cluster and 0 is left if they are in different
    clusters
for i in range(n):
    for j in range(n):
        if clusters_original[i] == clusters_original[j]:
            M[i][j] = 1


k2 = 3 # number of clusters
kmeans2 = KMeans(n_clusters=k2, random_state=0, n_init=10)
clusters_reduced = kmeans2.fit_predict(reduced_data) # the
    new clusters lables assigned
print("Unique clusters:\n",np.unique(clusters_original))
print("New cluster
    counts:\n",pd.Series(clusters_reduced).value_counts())

P = np.zeros((n, n))
# to populate P with 1 if observations at i and j are in the
    same cluster and 0 is left if they are in different
    clusters
for i in range(n):
    for j in range(n):
        if clusters_reduced[i] == clusters_reduced[j]:
            P[i][j] = 1

# checking the number of matching cluster assignments from M
    and P
matches = np.sum(M == P)
print("Number of matching cluster assignments between M and
    P:", matches)
```

```
print("Percentage␣of␣matching␣cluster␣assignments␣=␣",
    round(matches/(n*n) * 100,2))

testingfunc(data, clusters_original,
    kmeans1.cluster_centers_, k1)
testingfunc(reduced_data, clusters_reduced,
    kmeans2.cluster_centers_, k2)
ars = adjusted_rand_score(clusters_original,
    clusters_reduced) # comparing the two clusterings
print("Adjusted␣Rand␣Score:", ars)

# to visualize the average silhoutte score for the kmeans
    model instantiated using yellowbrick
visualizer = SilhouetteVisualizer(kmeans2)
visualizer.fit(reduced_data)
visualizer.show()
plt.show()

# 3D plots
fig = plt.figure(figsize=(12, 6))
# plotting the data points for the first 3 features
ax1 = fig.add_subplot(121, projection='3d')
ax1.scatter(data.iloc[:, 0], data.iloc[:, 1], data.iloc[:,
    2], c=clusters_original, cmap='viridis')
ax1.set_title("Original␣Data␣Distribution")
ax1.set_xlabel("Feature␣1")
ax1.set_ylabel("Feature␣2")
ax1.set_zlabel("Feature␣3")

# plotting the data points in the reduced dataset for the
    first 3 principal compenents
ax2 = fig.add_subplot(122, projection='3d')
ax2.scatter(reduced_data.iloc[:, 0], reduced_data.iloc[:,
    1], reduced_data.iloc[:, 2], c=clusters_reduced,
    cmap='viridis')
ax2.set_title("Reduced-Dimensional␣Data␣Distribution")
ax2.set_xlabel("Principal␣Component␣1")
ax2.set_ylabel("Principal␣Component␣2")
ax2.set_zlabel("Principal␣Component␣3")

# Elbow Method and Silhoutte Score Plot
# to find the optimal number of clusters (k), the elbow
    method is used and the silhoutte score for each is
    compared
silhouette = [] # for the silhoutte score
k = range(3,20) # clusters values from 4-20 are checked
```

```
# for each number of cluster, k-means is run and and inertia
    is calculated
# low inertia is preferred
for ks in k :
  kmeans = KMeans(n_clusters=ks, n_init=10, random_state=0)
  kmeans.fit(tsne)
  silhouette.append(silhouette_score(tsne, kmeans.labels_))

# to check for k with highest silhoutte score
plt.plot(k, silhouette)
plt.xlabel('K')
plt.ylabel('Silhouette␣score')
plt.title('Silhouette␣Analysis')
plt.grid(True)
plt.show()

k = 3 # number of clusters
# K-means clustering on the t-sne-reuced data
kmeans = KMeans(n_clusters=k, random_state=0, n_init=10)
clusters = kmeans.fit_predict(tsne) # clusters predcicted
    for the data
silhouette_tsne = silhouette_score(tsne, clusters) #
    silhoutte score of tsne reduced data
print("Silhouette␣score:␣", silhouette_tsne)

# Clustering visualization on t-SNE-reduced data
plt.scatter(tsne.iloc[:, 0], tsne.iloc[:, 1], c=clusters,
    cmap='viridis') # both the tsne dimensions
plt.scatter(kmeans.cluster_centers_[:, 0],
    kmeans.cluster_centers_[:, 1], c='red', marker='X',
    label='Cluster␣Centers') # for the cluster centres
plt.title('t-SNE␣plot␣with␣clusters')
plt.xlabel('t-SNE␣Dimension␣1')
plt.ylabel('t-SNE␣Dimension␣2')
plt.show()

T = np.zeros((n, n))
# to populate P with 1 if observations at i and j are in the
    same cluster and 0 is left if they are in different
    clusters
for i in range(n):
    for j in range(n):
        if clusters[i] == clusters[j]:
            T[i][j] = 1

# checking the number of matching cluster assignments from M
    and T
```

```
matches = np.sum(M == T)
print("Number␣of␣matching␣cluster␣assignments␣between␣M␣and␣
    T:", matches)
print("Percentage␣of␣matching␣cluster␣assignments␣=␣",
    round(matches/(n*n) * 100,2))

k = 4 # number of clusters
# K-means clustering on the t-sne-reuced data
kmeans = KMeans(n_clusters=k, random_state=0, n_init=10)
clusters = kmeans.fit_predict(tsne) # clusters predcicted
    for the data
silhouette_tsne = silhouette_score(tsne, clusters) #
    silhoutte score of tsne reduced data
print("\n\nSilhouette␣score:␣", silhouette_tsne)

testingfunc(tsne, clusters, kmeans.cluster_centers_, k)
ars = adjusted_rand_score(clusters_original, clusters) #
    comparing the two clusterings
print("Adjusted␣Rand␣Score:", ars)

"""### Impact of Initialization on K-Means Clustering"""

# some initializations/assignments are repeated in every
    section for better clarity in understanding the values
k = 3 # number of clusters
M_list = [] # to store matrices M1-M10
n = data.shape[0] # number of observations
randoms = [random.randint(1, 50) for _ in range(10)] #
    random initializations

# performing k-means clustering on the original dataset ten
    times with different initializations
for i in randoms:
    kmeans = KMeans(n_clusters=k, n_init=10, random_state=i,
        init='random') # k-means run with different random
        initial states
    kmeans.fit(data)
    clusters = kmeans.labels_

    # matrix M for this run is created as described in the
        previous section
    M = np.zeros((n,n))
    for i in range(n):
      for j in range(n):
        M[i,j] = 1 if clusters[i] == clusters[j] else 0
```

```
    M_list.append(M) # storing each of the M matrices made
       in each of the iterations

# if pairs of observations are in the same cluster in at
   least one run or not
# element-wise logical OR is done by rows
Mi = np.logical_or.reduce(M_list)

# if pairs of observations are in the same cluster in all
   ten runs or not
M_all = np.logical_and.reduce(M_list)

# percentage of pairs in the same cluster in all ten runs
percentage = (np.sum(M_all) / np.sum(Mi)) * 100

print("The␣percentage␣of␣pairs␣in␣the␣same␣cluster␣in␣all␣
   ten␣runs=␣", percentage, "%")

k = 3 # number of clusters
M_list = [] # to store matrices M1-M10
n = data.shape[0] # number of observations
randoms = [random.randint(1, 50) for _ in range(10)] #
   random initializations

# performing k-means clustering on the original dataset ten
   times with different initializations
for i in randoms:
    kmeans = KMeans(n_clusters=k, n_init=10, random_state=i,
       init='random') # k-means run with different random
       initial states
    kmeans.fit(reduced_data)
    clusters = kmeans.labels_

    # matrix M for this run is created as described in the
       previous section
    M = np.zeros((n,n))
    for i in range(n):
      for j in range(n):
        M[i,j] = 1 if clusters[i] == clusters[j] else 0

    M_list.append(M) # storing each of the M matrices made
       in each of the iterations

# if pairs of observations are in the same cluster in at
   least one run or not
# element-wise logical OR is done by rows
Mi = np.logical_or.reduce(M_list)
```

```
# if pairs of observations are in the same cluster in all
    ten runs or not
M_all = np.logical_and.reduce(M_list)

# percentage of pairs in the same cluster in all ten runs
percentage = (np.sum(M_all) / np.sum(Mi)) * 100

print("The␣percentage␣of␣pairs␣in␣the␣same␣cluster␣in␣all␣
    ten␣runs=␣", percentage, "%")

"""## Single-Linkage Clustering"""

k = 3 # number of clusters
# linkage matrix describes the hierarchical relationships
    between the data samples
linked = linkage(reduced_data, method='single') #
    single-linkage clustering on the reduced-dimensionality
    dataset
# returns an array of cluster labels for the observations
single_linkage_labels = fcluster(linked, k,
    criterion='maxclust') # use the maximum number of
    clusters specified by k

# matrix for the single-linkage clustering result
n = reduced_data.shape[0]
S = np.zeros((n, n))
for i in range(n):
  for j in range(n):
    if single_linkage_labels[i] == single_linkage_labels[j]:
      S[i,j] = 1

# Compare entries of S with P
matches = np.sum(S == P)
print("Dimensions␣of␣linkage␣matrix:␣", linked.shape)
print("The␣number␣of␣entries␣S␣has␣in␣common␣with␣P:",
    matches)
print("\nPercentage␣of␣matching␣cluster␣assignments␣=␣",
    round(matches/(n*n) * 100,2))

# silhoutte score of the single linkage clustering on the
    reduced-dimensionality dataset
silhouette_reduced_linkage = silhouette_score(reduced_data,
    single_linkage_labels)
print("Silhouette␣coefficient:", silhouette_reduced_linkage)
```

```
# Scatterplots with the first two prinicpal components on
    the axes
# for k-means clustering
sns.scatterplot(x=reduced_data.iloc[:, 0],
    y=reduced_data.iloc[:, 1], c=clusters_reduced,
    cmap='viridis')
plt.scatter(kmeans2.cluster_centers_[:, 0],
    kmeans2.cluster_centers_[:, 1], c='red', marker='X',
    label='Cluster Centers') # for the cluster centres
plt.title('k-Means Clustering')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()

# for single-linkage clustering
sns.scatterplot(x=reduced_data.iloc[:, 0],
    y=reduced_data.iloc[:, 1], c=single_linkage_labels,
    cmap='viridis')
plt.title('Single-Linkage Clustering')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()

def single_silhouette(data1):
  '''
    For the Silhoutte Score Plot to find optimal k
  '''
  # to find the optimal number of clusters (k), the
     silhoutte score for each is compared
  silhouette = [] # for the silhoutte score
  k = range(3,20) # clusters values from 4-20 are checked
  # for each number of cluster, single linkage clustering is
     run and and the score is calculated
  for ks in k :
    linked = linkage(data1, method='single') #
       single-linkage clustering on the
       reduced-dimensionality dataset
    single_linkage_labels = fcluster(linked, ks,
       criterion='maxclust') # use the maximum number of
       clusters specified by k
    silhouette.append(silhouette_score(data1,
       single_linkage_labels))

  # to check for k with highest silhoutte score
  plt.plot(k, silhouette)
  plt.xlabel('K')
  plt.ylabel('Silhouette score')
```

```
  plt.title('Silhouette Analysis')
  plt.grid(True)
  plt.show()

single_silhouette(reduced_data)

# Dendrogram for single-linkage clustering as its a
    hierarchial clustering algorithm
plt.figure(figsize=(85,25))
dendrogram(Z=linked)
plt.title('Dendrogram for Single-Linkage Clustering')
plt.ylabel('Distance')
plt.show()

single_silhouette(data)

k = 3
# linkage matrix describes the hierarchical relationships
    between the data samples
linked = linkage(data, method='single') # single-linkage
    clustering on the dataset
# returns an array of cluster labels for the observations
single_linkage_labels1 = fcluster(linked, k,
    criterion='maxclust') # use the maximum number of
    clusters specified by k

# silhoutte score of the single linkage clustering on the
    dataset
silhouette_linkage1 = silhouette_score(data,
    single_linkage_labels1)
print("Silhouette coefficient:", silhouette_linkage1)

single_silhouette(tsne) # for tsne data

# linkage matrix describes the hierarchical relationships
    between the data samples
k = 3
linked = linkage(tsne, method='single') # single-linkage
    clustering on the reduced-dimensionality dataset
# returns an array of cluster labels for the observations
single_linkage_labels2 = fcluster(linked, k,
    criterion='maxclust') # use the maximum number of
    clusters specified by k

# silhoutte score of the single linkage clustering on the
    reduced-dimensionality dataset
```

```
silhouette_reduced_linkage2 = silhouette_score(tsne,
    single_linkage_labels2)
print("Silhouette coefficient:", silhouette_reduced_linkage2)

"""## GMM + Expectation-Maximization Algorithm"""

def gmm(data1, k):
  '''
    To perform expectation-maximization algorithm with k
       clusters on reduced-dimensional data
    and return the proababilities and labels
  '''
  model = GaussianMixture(n_components=k,
     init_params='k-means++', covariance_type="full") #
     instantiating a GMM with the given parameters
  model.fit(data1) # fitting the model on the data reduced
     by PCA

  # Probabilistic assignment of observations to clusters
  # components' density for each sample in the dataset
  probs = model.predict_proba(data1)
  # cluster centres generated
  gmm_labels = model.predict(data1)

  # printing the AIC and BIC scores
  print("AIC Score:", model.aic(data1))
  print("BIC Score:", model.bic(data1))
  print("Silhouette coefficient:", silhouette_score(data1,
     gmm_labels))


  #
     ------------------------------------------------------------------------
  # for confidence ellipses- following code modified
  # from https://scikit-learn.org/stable/auto_examples/
  # mixture/plot_gmm_selection.html
  colors = sns.color_palette("bright", k) # to set the
     colors for plotting
  fig, ax = plt.subplots() # figure and axis for plotting

  # iterating over the means, covariances, and colors to
     create ellipses
  # based on the mean and covariance of differenr colors
     around the clusters
  for i, (mean, cov, color) in enumerate(zip(model.means_,
     model.covariances_, colors)):
```

```
        eig_vals, eig_vector = linalg.eigh(cov) # eigenvalues
            and eigenvectors of the covariance matrix

        # to check if any data points belong to the current
            component/cluster
        if not np.any(gmm_labels == i):
            continue

        # scatterplot of the data points belonging to the
            current component/cluster
        ax.scatter(data1.iloc[gmm_labels == i, 0],
            data1.iloc[gmm_labels == i, 1], s=0.8, color=color)

        angle = np.arctan2(eig_vector[0][1], eig_vector[0][0])
            # represents the angle of rotation for the ellipse
        angle = 180.0 * angle / np.pi  # converting the angle
            from radians to degrees
        v = 2.0 * np.sqrt(2.0) * np.sqrt(eig_vals) # lengths
            of the major and minor axes of the ellipse

        # Ellipse object with the parameters calculated for
            the cluster
        ellipse = Ellipse(mean, eig_vals[0], eig_vals[1],
            angle=180.0 + angle, color=color, alpha=0.5)
        ax.add_artist(ellipse) # the ellipse is added to the
            plot

    ax.set_title(f"GMM␣with␣{k}␣number␣of␣components")
    plt.show()

    return probs, gmm_labels

# GMM on the data reduced by PCA
# Default covariance type where each component has its own
    general covariance matrix.
probs, gmm_labels = gmm(reduced_data,3)
n = len(reduced_data) # number of observations
E = np.zeros((n,n)) # instantiating matrix-E of size nxn

# to loop over each pairs of observations
for i in range(n):
    for j in range(n):
        # element-wise multiplication of the probability
            vectors
        # of observations i and j are summed
         E[i, j] = np.sum(probs[i] * probs[j])
```

```
# to ensure entries of E are between 0 and 1
E = np.clip(E, 0, 1)

#-------------------------------------------------------------------------------
# represents number of observations with at least two
   different clusters
count = 0

for i in range(n):
    # indices of clusters with probability > 1% for an
       observation
    clusters_above_threshold = np.where(probs[i] > 0.01)[0]

    # to check if there are at least two different clusters
    if len(set(clusters_above_threshold)) >= 2:
        count += 1

print(f"\n\nPercentage␣of␣observations␣with␣at␣least␣two␣
   different␣clusters:␣{((count/n)*100):.2f}%")

# Histogram to show the bell-shaped distribution of the data
plt.hist(reduced_data, bins=50)
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.show()

# inspired from-
#https://scikit-learn.org/stable/auto_examples/mixture/plot_gmm_selection.
#html#:~:text=In%20this%20case%2C%20both%20the,among%20a%20set%20of%20candidates

def gmm_bic_score(estimator, X):
  '''
    For GridSearchCV; to use BIC to select parameters
  '''
  return -estimator.bic(X) # negative as GridSearchCV
     expects a score to maximize

def bic_grid_search(data1):
  '''
    To try Grid Search over the parameters using BIC score
    lower the BIC, the better
  '''
  parameters = {'n_components': [3,4,5,6,7,8,9,10,11]} #
     parameter to search for
  gscv =
     GridSearchCV(GaussianMixture(init_params='k-means++'),
     parameters, scoring=gmm_bic_score) # to search over the
```

```
      parameter for the GMM
  gscv.fit(data1) # to fit the model on the PCA-reduced
      dataset
  test_score = - (gscv.cv_results_['mean_test_score'])

  sns.barplot(x=gscv.cv_results_['param_n_components'],
      y=test_score) #optional
      hue=gscv.cv_results_['param_covariance_type']
  plt.xlabel('Number␣of␣Components')
  plt.ylabel('BIC␣Score')
  plt.show()

bic_grid_search(reduced_data)

probs, gmm_labels = gmm(reduced_data, 3) # GMM on
    PCA-reduced data with optimal values, only tied and full
    work
print("Unique␣clusters:␣", np.unique(gmm_labels))

bic_grid_search(data)

probs, gmm_labels = gmm(data, 3) # GMM on the original data
    with optimal values
print("Unique␣clusters:␣", np.unique(gmm_labels))

bic_grid_search(tsne)

# for GMM with t-sne data
probs, gmm_labels = gmm(tsne,3) # GMM on tsne-reduced data
    with optimal values
print("Unique␣clusters:␣", np.unique(gmm_labels))

"""## Using DBSCAN"""

def dbscan(data1, eps):
  '''
    Running DBSCAN with optimal parameters on the dataframe
       given
  '''
  # parameter values to test
  params_grid = {
      'eps': eps,
      'min_samples': [3, 5, 8, 9, 10, 15, 20, 30]
  }

  score = -1 # the best silhoutte score
```

```
  best_params = { # best set of parameters, initialized with
     fairly suitable values
      'eps': eps[0],
      'min_samples': 10
  }

  for params in ParameterGrid(params_grid): # to iterate and
     test all possible combinations of the values
      model = DBSCAN(**params) # created a DBSCAN model with
         the parameters given
      labels = model.fit_predict(data1) # fitting the model
         on the data

      # to check if there are at least two unique labels for
         the silhouette_score() function to work
      if len(np.unique(labels)) > 1:
          sscore = silhouette_score(data1, labels) #
             silhouette score
          if sscore > score: # if the new score is better,
             the new set of parameters are better
              score = sscore
              best_params = params

  print("Best set of parameters:", best_params)
  print("Best calculated silhouette score:", score)

  # running DBSCAN with optimal parameters
  model = DBSCAN(eps = best_params['eps'], min_samples =
     best_params['min_samples'])
  labels = model.fit_predict(data1) # fitting the model on
     the given data
  print('Number of clusters: ', np.unique(labels))
  # scatter plot with 2 features only, for DBSCAN clusters
  plt.scatter(data1.iloc[:, 0], data1.iloc[:, 1], c=labels,
     cmap='viridis')
  plt.title('DBSCAN Clustering')
  plt.colorbar()
  plt.show()

# to get the minimum and maximum eps value to check from the
   graph, the point of consistent increase in the y-axis
knn = NearestNeighbors(n_neighbors=5).fit(data) # to create
   an instance of NearestNeighbors with k=5 and fit it to
   the original data
distances, indices = knn.kneighbors(data) # to compute the
   distances and indices of the k nearest neighbors for each
   data point
```

```
distances = np.sort(distances, axis=0) # to sort the
    distances along the rows to obtain distances in ascending
    order for each data point
plt.plot(distances[:, 1]) # to plot the distances to the
    second nearest neighbor for each data point
plt.show()

dbscan(data, np.arange(8, 25, 2)) # eps is values in range
    8-25 at every 2 step

# to get the minimum and maximum eps value to check from the
    graph, the point of consistent increase in the y-axis
knn = NearestNeighbors(n_neighbors=5).fit(reduced_data) # to
    create an instance of NearestNeighbors with k=5 and fit
    it to the original data
distances, indices = knn.kneighbors(reduced_data) # to
    compute the distances and indices of the k nearest
    neighbors for each data point
distances = np.sort(distances, axis=0) # to sort the
    distances along the rows to obtain distances in ascending
    order for each data point
plt.plot(distances[:, 1])
plt.show()

dbscan(reduced_data, np.arange(1.2,5,0.1)) # for reduced
    dataset

# to get the minimum and maximum eps value to check from the
    graph, the point of consistent increase in the y-axis
knn = NearestNeighbors(n_neighbors=5).fit(tsne) # to create
    an instance of NearestNeighbors with k=5 and fit it to
    the original data
distances, indices = knn.kneighbors(tsne) # to compute the
    distances and indices of the k nearest neighbors for each
    data point
distances = np.sort(distances, axis=0) # to sort the
    distances along the rows to obtain distances in ascending
    order for each data point
plt.plot(distances[:, 1])
plt.show()

dbscan(tsne, np.arange(1.3,2.5,0.025)) # for the
    tsne-reduced model

"""## Using BIRCH"""

def birch(data1):
```

```
'''
  Running BIRCH with optimal parameters on the dataframe
     given
'''
# grid of suitable parameter values to test for
params_grid = {
    'threshold': [0.01, 0.05, 0.1, 0.5, 1.0, 1.5, 2.0],
    'n_clusters' : [1,2,3,4,5,6,7,8,9,10],
    'branching_factor': [3, 4, 5, 10, 20, 30, 40]
}

score = -1 # the best silhoutte score
best_params = { # best set of parameters, initialized with
   fairly suitable values
    'threshold': 0.1,
    'n_clusters' : 3,
    'branching_factor': 5
}

for params in ParameterGrid(params_grid): # to iterate and
   test all possible combinations of the values
    model = Birch(**params) # created a BIRCH model with
       the parameters given
    labels = model.fit_predict(data1) # fitting the model
       on the data

    # to check if there are at least two unique labels for
       the silhouette_score() function to work
    if len(np.unique(labels)) > 1:
        sscore = silhouette_score(data1, labels) #
           silhouette score
        if sscore > score: # if the new score is better,
           the new set of parameters are better
            score = sscore
            best_params = params

print("Best set of parameters:", best_params)
print("Best calculated silhouette score:", score)

# running the model with optimal parameters
model = Birch(branching_factor =
   best_params['branching_factor'], n_clusters =
   best_params['n_clusters'], threshold =
   best_params['threshold'])
labels = model.fit_predict(data1) # assign each data point
   to a cluster and output labels
print('Number of clusters: ', np.unique(labels))
```

```
  # scatter plot with 2 features only, for BIRCH clusters
  plt.scatter(data1.iloc[:, 0], data1.iloc[:, 1], c=labels,
     cmap='viridis')
  plt.title('BIRCH Clustering')
  plt.xlabel('Feature 1')
  plt.ylabel('Feature 2')
  plt.show()

birch(data) # BIRCH on the original dataset

birch(reduced_data) # BIRCH on the PCA-reduced dataset

birch(tsne) # BIRCH on the t-SNE-reduced dataset
```

References

[1] G. Seif. The 5 clustering algorithms data scientists need to know. [Online]. Available: https://towardsdatascience.com/the-5-clustering-algorithms-data-scientists-need-to-know-a36d136ef68

[2] What is machine learning? definition, types, tools & more. [Online]. Available: https://www.datacamp.com/blog/what-is-machine-learning

[3] The curse of dimensionality in machine learning: Challenges, impacts, and solutions. [Online]. Available: https://www.datacamp.com/blog/curse-of-dimensionality-machine-learning

[4] Basic guide to scikit-learn - coding ninjas. [Online]. Available: https://www.codingninjas.com/studio/library/basic-guide-to-scikit-learn

[5] scikit-learn: machine learning in python — scikit-learn 1.3.2 documentation. [Online]. Available: https://scikit-learn.org/stable/

[6] SciPy. [Online]. Available: https://scipy.org/

[7] C. GOYAL. Complete guide to expectation-maximization algorithm. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/05/complete-guide-to-expectation-maximization-algorithm/

[8] Clustering algorithms | machine learning. [Online]. Available: https://developers.google.com/machine-learning/clustering/clustering-algorithms

[9] A. Singh. Build better and accurate clusters with gaussian mixture models. [Online]. Available: https://www.analyticsvidhya.com/blog/2019/10/gaussian-mixture-models-clustering/

[10] S. Ghosh. The ultimate guide to evaluation and selection of models in ML. [Online]. Available: https://neptune.ai/blog/ml-model-evaluation-and-selection

[11] J. Brownlee. 10 clustering algorithms with python. [Online]. Available: https://machinelearningmastery.com/clustering-algorithms-with-python/

[12] Creating animation to show 4 centroid-based clustering algorithms using python and sklearn | by boriharn k | towards data science. [Online]. Available: https://towardsdatascience.com/creating-animation-to-show-4-centroid-based-clustering-algorithms-using-python-and-sklearn-d397ade89cb3

[13] Z. Bar-Joseph, D. K. Gifford, and T. S. Jaakkola, "Fast optimal leaf ordering for hierarchical clustering," vol. 17, pp. S22–S29. [Online]. Available: https://academic.oup.com/bioinformatics/article/17/suppl_1/S22/261423

[14] A. Sharma. How to master the popular DBSCAN clustering algorithm for machine learning. [Online]. Available: https://www.analyticsvidhya.com/blog/2020/09/how-dbscan-clustering-works/

[15] 8 clustering algorithms in machine learning that all data scientists should know. [Online]. Available: https://www.freecodecamp.org/news/8-clustering-algorithms-in-machine-learning-that-all-data-scientists-should-know/

[16] Exploring clustering methods in machine learning. [Online]. Available: https://www.opensourceforu.com/2019/12/exploring-clustering-methods-in-machine-learning/

[17] N. J. Benzer. BALANCED ITERATIVE REDUCING AND CLUSTERING USING HEIRARCHIES(BIRCH). [Online]. Available: https://medium.com/@noel.cs21/balanced-iterative-reducing-and-clustering-using-heirachies-birch-5680adffaa58

[18] t-SNE clearly explained. an intuitive explanation of t-SNE... | by kemal erdem (burnpiro) | towards data science. [Online]. Available: https://towardsdatascience.com/t-sne-clearly-explained-d84c537f53a

[19] Introduction to t-SNE in python with scikit-learn. [Online]. Available: https://danielmuellerkomorowska.com/2021/01/05/introduction-to-t-sne-in-python-with-scikit-learn/

[20] pandas - python data analysis library. [Online]. Available: https://pandas.pydata.org/

[21] J. Wohlenberg. 3 versions of k-means. [Online]. Available: https://towardsdatascience.com/three-versions-of-k-means-cf939b65f4ea

[22] Gaussian mixture model selection. [Online]. Available: https://scikit-learn/stable/auto_examples/mixture/plot_gmm_selection.html

[23] Bayesian information criterion (BIC) - data analysis expert in new york, chicago, san francisco, boston, los angeles. [Online]. Available: http://stanfordphd.com/BIC.html

[24] A. Sharma and A. Sharma, "KNN-DBSCAN: Using k-nearest neighbor information for parameter-free density based clustering," in *2017 International Conference on Intelligent Computing, Instrumentation and Control Technologies (ICICICT)*, pp. 787–792. [Online]. Available: https://ieeexplore.ieee.org/document/8342664

[25] C. Martins. Understanding BIRCH clustering: Hands-on with scikit-learn. [Online]. Available: https://medium.com/codex/understanding-birch-clustering-hands-on-with-scikit-learn-c84d86e8f66b

[26] H. Gültekin. What is silhouette score? [Online]. Available: https://medium.com/@hazallgultekin/what-is-silhouette-score-f428fb39bf9a

[27] Silhouette plot - MATLAB silhouette. [Online]. Available: https://www.mathworks.com/help/stats/silhouette.html

[28] C. Ellis. When to use t-sne. [Online]. Available: https://crunchingthedata.com/when-to-use-t-sne/

[29] Papers with code - ensemble clustering explained. [Online]. Available: https://paperswithcode.com/method/ensemble-clustering

[30] Explain BIRCH algorithm with example. [Online]. Available: https://www.ques10.com/p/9298/explain-birch-algorithm-with-example/