Machine Learning Applied to Fluorescence Microscopy Images to Assist the Development of Malaria Transmission - Blocking Drugs

Vishanth Suresh
201669090

Supervised by Dr.Sofya Titarenko

Submitted in accordance with the requirements for the
module MATH5872M: Dissertation in Data Science and Analytics
as part of the degree of

Master of Science in Data Science and Analytics

The University of Leeds, School of Mathematics

September 2024

The candidate confirms that the work submitted is his/her own and that appropriate credit has been given where reference has been made to the work of others.

UNIVERSITY OF LEEDS

School of Mathematics

Declaration of Academic Integrity for Individual Pieces of Work

I declare that I am aware that as a member of the University community at the University of Leeds I have committed to working with Academic Integrity and that this means that my work must be a true expression of my own understanding and ideas, giving credit to others where their work contributes to mine.

I declare that the attached submission is my own work.

Where the work of others has contributed to my work, I have given full acknowledgement using the appropriate referencing conventions for my programme of study.

I confirm that the attached submission has not been submitted for marks or credits in a different module or for a different qualification or completed prior to entry to the University.

I have read and understood the University's rules on Academic Misconduct. I know that if I commit an academic misconduct offence there can be serious disciplinary consequences.

I re-confirm my consent to the University copying and distributing any or all of my work in any form and using third parties to verify that this is my own work, and for quality assurance purposes.

I confirm that I have declared all mitigating circumstances that may be relevant to the assessment of this piece of work and I wish to have taken into account.

Student Signature:
Vishnath

Student Number:2016669090

Student Name: **Date:02/09/2024**

Vishanth Suresh

Please note:

When you become a registered student of the University at first and any subsequent registration you sign the following authorisation and declaration:

"I confirm that the information I have given on this form is correct. I agree to observe the provisions of the University's Charter, Statutes, Ordinances, Regulations and Codes of Practice for the time being in force. I know that it is my responsibility to be aware of their contents and that I can read them on the University web site. I acknowledge my obligation under the Payment of Fees Section in the Handbook to pay all charges to the University on demand.

I agree to the University processing my personal data (including sensitive data) in accordance with its Code of

Practice on Data Protection http://www.leeds.ac.uk/dpa . I consent to the University making available to third parties (who may be based outside the European Economic Area) any of my work in any form for standards and monitoring purposes including verifying the absence of plagiarised material. I agree that third parties may retain copies of my work for these purposes on the understanding that the third party will not disclose my identity.'"

Abstract

This dissertation investigates the PhIDDLI(Phenotype Image Data and Digital Learning Innovation) pipeline, a real-time biomedical research analysis pipeline, and provides improvised approach to derive better insights from the biomedical images. The pipeline is used to study the blocking properties of the drugs used to stop the transmission of malaria which is one of the deadliest diseases in the world. The PhIDDLI pipeline is covered in this dissertation, along with its uses and disadvantages. In order to get better study insights, various modeling experiments have been carried out and an improvised model has been built. An introduction to the problem understanding is included in Chapter 1. The background survey of the dataset on the spread of malaria and the function of medications that prevent transmission is explained in Chapter 2. The evolution of the Plasmodium gametocyte cells, which are in charge of malaria transmission, is also covered in this chapter.
The literature that is needed to conduct the experiments and enhance the pipeline model is included in Chapter 3. In Chapter 4, the pipeline's steps are examined, and the model is diagnosed using the necessary metrics. Additionally, it talks about the customized CNN model that is used to verify the cluster's purity. In Chapter 5, it is discussed how modeling high dimensional datasets using a tiny portion of the pipeline's dataset presents challenges. In Chapter 6, clustering studies using K-Means and Hierarchical clustering techniques are covered. Both Isomap and UMAP data points are used for each method. The use of multi-attribute methods to establish benchmark scores for each model and subsequently choose the best model is covered in Chapter 7. In Chapter 8, it is discussed how to apply semi-supervised fuzzy clustering to the chosen model in order to extract fuzzy and deep patterns that may help researchers and biologists create medications that impede transmission. Chapter 9 concludes by discussing future research directions to improve the pipeline and the actions taken to obtain more insightful knowledge from it.

Acknowledgements

I want to express my gratitude to Dr. Sofya Titarenko, my dissertation supervisor, for always pointing me in the correct direction. In order to finish this dissertation, I would also like to extend my gratitude to Dr. Michael Delves from the London School of Hygiene for his support and advice on biomedical matters. Additionally, I would like to thank University of Leeds, specifically the School of Mathematics, for their excellent instruction, guidance, and support during the course. Lastly, I want to express my gratitude to my parents for their emotional and financial assistance.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

An unsupervised machine learning technique called clustering is used to analyze unlabeled data that contains hidden patterns(MacQueen 1967). Dimensionality reduction helps to analyse,visualise and interpret high dimensional data points(Ghodsi 2006). There are several approaches to dimensionality reduction and clustering. The objective is to investigate the approaches using a real-time application, discover how each model fits the dataset, and select the best match.
The PhIDDLI (Phenotype Image Data and Digital Learning Innovation) pipeline, which categorizes fluorescent microscopic pictures of Plasmodium Falciparum gametocyte cells responsible for malaria transmission, is the real-time application selected for this dissertation(Tsebriy et al. 2023). Malaria is one of the most serious illnesses on the world that is brought on by mosquito bites. Malaria causes symptoms such as cough, jaundice, anemia, renal failure, and ultimately death if treatment is delayed(Organization 2000).

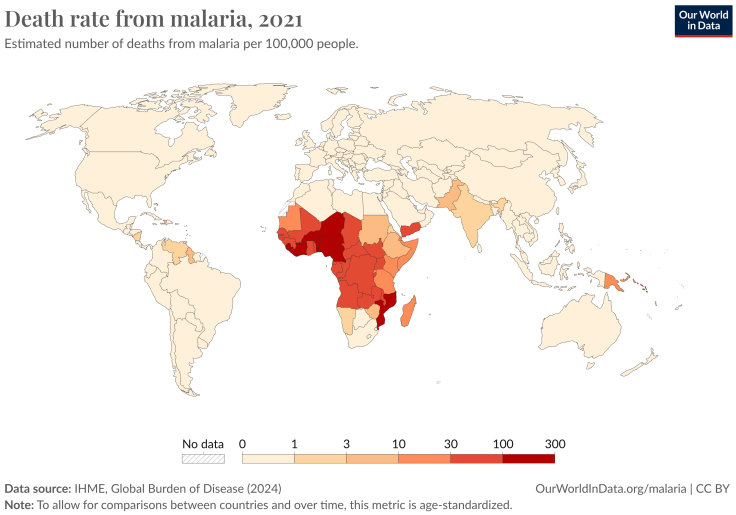


Figure 1.1: Number of malarial deaths per 100000 people. Source:`https://ourworldindata.org/malaria` .

This pipeline's main objective is to learn more about the molecules' blocking abilities,

which can stop the disease from spreading. Based on the findings, compounds exhibiting different blocking characteristics may be taken into consideration for the creation of TBDs, or transmission-blocking medications, which can halt the spread of malaria to further human hosts. Different transmission inhibiting compounds are allowed to react with Plasmodium Falciparum gametocyte cells for a full day. The resulting reactions are photographed under a microscope and subsequently processed through a pipeline for analysis and categorization(Tsebriy et al. 2023). This dissertation's main objective is to analyze the pipeline, identify areas for improvement, run a number of different experiments, and track the models' responses. This leads to enhancement in analysing the hidden and deep patterns by the biologists and researchers for better drug development.

1.1 Review of the Dissertation

In this dissertation, the following procedures are used to undertake alternative clustering experiments to the PhIDDLI model.

- Examination of the PhIDDLI pipeline, detailing how unprocessed microscopic images are grouped based on their blocking characteristics.
- Model diagnosis, including a discussion of its shortcomings and an efficiency assessment using the required metrics.
- Explanation of the alternative experiments conducted, along with their observations and assessment metrics.
- Selection of a superior model using a multi-attribute approach that assigns a benchmark score to each model.
- Upgrading the chosen model with respect of the key findings and observations.

Chapter 2

Background Information of the Dataset

2.1 Life cycle of Malaria

Plasmodium Falciparum, a parasite that causes malaria, is spread via the bite of female Anopheles mosquitoes(World Health Organization 2023). Humans serve as the hosts in the malaria life cycle, while mosquitoes act as the carriers(World Health Organization 2023).

2.1.1 Stage 1: Mosquitoes to Human

Sporozites, the infectious form of the parasite, are injected into the blood vessels by the bite. They reproduce asexually and impact the liver through the bloodstream. During this procedure, merozoites are produced. From the liver, these merozoites go throughout the bloodstream(World Health Organization 2023). The merozoites begin sexual reproduction in the circulation, which results in the development of gametocytes.

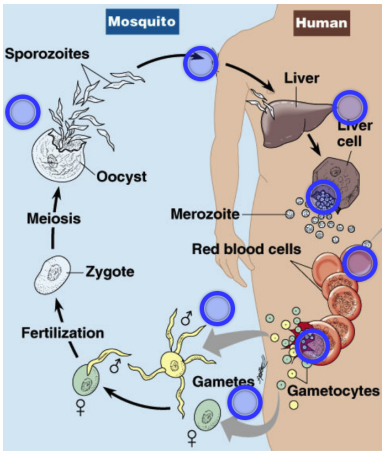


Figure 2.1: Lifecycle of Malaria(Quizlet 2024).

2.1.2 Stage 2: Humans to mosquitoes

After that, the gametocyte cells undergo exflagellation. Then the exflagellated cells enter the Anopheles mosquito during the blood meal. The male and female gametocytes combine to generate zygotes inside the mosquito's midgut. After developing into ookinetes, the zygotes pierce the midgut wall and eventually form oocysts. Sporozoites enter the salivary glands of mosquitoes when the oocyst bursts and are then injected back into human bloodstreams.

2.2 Role of Transmission Blocking Drugs

The goal of the transmission-blocking drugs is to arrest the gametocytes before exflagellation so that unactivated gametocytes cannot regenerate into sporozoites in the mosquito's midgut. Some of the TBDs from Tres Cantos Antimalarial set are TC01, TC02, TC880, etc.

2.3 The Primary Screen Test

A subset of transmission blocking molecules from the Tres Cantos Antimalarial Set (TCAMS)(Tsebriy et al. 2023) are used in "the primary screen test," which screens at 2μm against plasmodium gametocytes. The cells are photographed under a microscope after a 24-hour period, and those images form the primary screen dataset. Afterwards, the PhIDDLI pipeline processes the images in order to learn more about the drugs' blocking properties(Tsebriy et al. 2023).
The TIF(Tagged Image Format)-formatted microscopic images and the XML(Extensible Markup Language)-formatted bounds of the detected cells in the image make up the primary screen dataset. At the moment, the open-source ICY tool is used to manually identify the cells. Fluorescence microscopy is used to obtain the raw TIF pictures. Each image has a depth of 16 bits and has two channels. The dataset is 19.9 gigabytes in size.To access the primary screen dataset, visit the following link: Click here to access.

Chapter 3

Literature Review

3.1 Application Oriented Literature

3.1.1 Tsebriy et al. (2023)

The primary information on the pipeline is discussed in this paper. The setup of the pipeline to investigate the test molecules' response to Plasmodium falciparum gametocytes has been explained by the authors. One of the world's worst mosquito-borne diseases, malaria, requires the use of transmission-blocking drugs (TBDs), which the authors have researched. Their goal is to investigate the blocking properties of 40 TBDs through the use of a primary screen test, in which the drugs are administered to plasmodium gametocytes and the results are tracked for a full day. The images produced are then processed through a machine learning pipeline known as the PhIDDLI pipeline, which classifies the images into clusters for further investigation.

The pipeline's layers—cell detection, cell extraction, feature extraction, dimensionality reduction, clustering, and visualization—are further covered in the paper. It also goes over how the pictures are categorised according to how the drugs affect the gametocytes.

Consequently, the PhIDDLI pipeline produced clusters with varying blocking characteristics, such as gametocytes that arrested early, mid, and late. As a result, their related drugs are explained and provided. Additionally, the work improves our understanding of the blocking properties of molecules that prevent transmission. The paper provides an overview of the pipeline's processes and an early analysis of the clusters, which serves as the foundation for this dissertation.

3.1.2 Xu & Wunsch (2010)

In order to investigate the patterns in the dataset, the authors of this paper used a biological research project as a case study, applying a number of clustering approaches.The authors have provided an overview of the ways in which clustering algorithms can be

used in biomedical research to identify interesting trends and separate samples with similar features. In the beginning, they provided an overview of clustering approaches by determining the closest neighbors using a variety of metrics, including Pearson coefficient, Euclidean distance, Minkowski distance, and Cosine similarity. They have demonstrated, with the use of a real-time application, how the deployment of clustering techniques might enhance medical research.
They studied yeast cells at various stages of their cycle in a case study. Similar-looking cells are grouped together. The dataset's over 3000 cells are categorized into 30 distinct clusters. The authors have been able to investigate and analyze cells with new characteristics because to this.
Additionally, they have experimented with soft clustering methods that allow a data point to be assigned to numerous clusters simultaneously, such fuzzy C means(Xu & Wunsch 2010). By using this technique, multifunctional cells are identified, meaning that their characteristics can be assigned to multiple clusters, and their responses to various cellular processes can be investigated. In order to uncover hidden patterns in the data for the dissertation, this paper provided guidance on the application of hard and soft clustering algorithms for biomedical research.

3.2 Methodology Oriented Literature

3.2.1 Toma˘sev & Radovanovi´c (2016)

The PhIDDLI model has applied high dimensional clustering on the dataset. This paper has been used to investigate whether or not high dimensional clustering is causing the PhIDDLI model to become restricted. The authors have discussed about the challenges faced while clustering with high dimensional datasets. They explain about phenomenon like data sparsity, distance concentration and hubness in high dimensional datasets that affect the quality of the clusters. Moreover the paper illustrates how the clustering quality indexes like Silhouette score, Calinski Harabasz index and Davies Bouldin index gets unstable in high dimensional clustering.

The authors have demonstrated the difficulties in high dimensional clustering by creating datasets with several dimensions and distinct cluster sets. According to the authors' findings, certain quality indices exhibit stability across all dimensions, whereas other indexes exhibit instability when the dimensions get increased. Nevertheless, the outcomes are limited to artificial data.

3.2.2 Van Der Maaten & Hinton (2008)

The authors have introduced a new non linear dimensionality reduction called t-SNE (t-distributed Stochastic Neighbor Embedding). Since the PhIDDLI model visualizes the clusters using the t-SNE technique, it is necessary to understand the technique by

using this paper.The authors claim that this method can overcome overcrowding and optimization issues while reconstructing the low dimensional structure. The aim of this technique is to preserve most of the global and local structure of the dataset after reducing the dimensions.

- The first step of t-SNE is to calculate the pairwise distances between the data points using euclidean distances and turn them into conditional probabilities.
- The authors then calculate a similarity score using these probabilities and tries to move the similar points closer using a student t-distribution resulting in clearing overcrowding of data points.
- Then the conversion loss is calculated using KL-Divergence theorem and it is optimised using gradient descent.

The technique has been implemented on MNIST dataset and COIL-20 dataset and the results are observed. The advantage of using this technique has been clearly visualized and compared with other techniques to highlight its superiority.

3.2.3 McInnes et al. (2018)

The Uniform Manifold Approximation and Projection, developed by McInnes, Healy, and Melville in this publication, is an excellent technique for dimensionality reduction. This is one of the non-linear dimensionality reduction methods that can reduce the dimensions without significantly sacrificing the dataset's global or local structure. As an alternate experiment to t-SNE, the UMAP technique will be utilized to examine if the data points are well separated and how well the clustering model functions with UMAP structure. This method has been selected for the dissertation as UMAP arrived in 2018 whereas t-SNE arrived in 2008. The authors claim that this approach functions more quickly and effectively than earlier methods such as PCA (Principle Component Analysis), MDS (Multi Dimensional Scaling), and t-SNE (t-distributed Stochastic Neighbour Embedding).

- The authors have provided an explanation of how the nearest neighbours are determined, and how the fuzzy graph is created by combining the neighbour data points with edges and their weights.
- Additionally, they have described the last optimisation strategy that takes the proper step-by-step approach to maintain the global and local structure even after the dimension reduction.
- The authors have demonstrated the benefits of the UMAP in terms of quicker and more effective computation by comparing its performance to that of other dimensionality reduction methods as they claimed in the beginning.

The authors have also provided an explanation of how UMAP is used in a variety of fields, including image processing, language processing, and bioinformatics. In order to comprehend its nature, they have also applied the technique on real-time RNA sequencing data. The sensitivity of its hyperparameters, such as nearest neighbors, minimum distance to pack points together, and number of features in low dimensional space, is also discussed in detail in the study.

3.2.4 The Greedy Choice (2017)

The online resource by The Greedy Choice has explained about the steps involved in the Isomap(ISOmetric MAPping) algorithm, which is one of the non linear dimensionality reduction techniques. In this dissertation, an additional experiment to t-SNE is the Isomap approach. Moreover Isomap is selected for experiment as it arrived later than t-SNE.

- The first step involves determining the nearest neighbours for each data point. It is done with the help of k value by which the number of neighbours are decided.
- The euclidean distance is calculated for pairwise points and a neighbourhood graph is calculated.
- Now the shortest path between all pair of nodes are calculated using Floyd-Warshall algorithm.
- Then the final distance matrix is applied for eigen decomposition to determine the low dimensional datapoints.

The resource states that Isomap can be efficiently used for image processing, speech analysis and biological datasets. Overall it efficiently covers the fundamental concepts of the algorithm and its application in real world.

3.2.5 MacQueen (1967)

John MacQueen in 1967 introduces a method to partition data points into clusters called the k-means clustering. Both the PhIDDLI pipeline and the alternate experiments for this dissertation use this clustering technique. Hence this paper is used to understand and apply the corresponding theory. This paper is the foundation to understand the concept of clustering in a statistical approach. This method made data analysis pretty much easier as it can classify the data points in an unsupervised way.

- The process starts with initialising cluster centers. At first they are assigned randomly.
- The nearest points to the centers are calculated using the euclidean distance.

- Now the mean for each cluster is calculated and it is assigned as the new center.
- The above steps are repeated until convergence,i.e, until the clusters no longer change.

The methodology is clearly explained with mathematical equations along with proper calculations. The author has applied the technique on synthetic data as well as iris dataset which is a real world dataset. Overall the paper helps data analysis and research in enormous ways and continues to influence the understanding of real world problems.

3.2.6 Murtagh & Contreras (2012)

Authors Murtagh and Contreras have explained about their experiments on hierarchical clustering algorithms by implementing them on synthetic datasets with appropriate theories. In this dissertation, the findings of the K-means model are compared with the clustering technique to see which model best fits the dataset among the alternative experiments.

- Agglomerative Hierarchical Clustering includes methods like single,complete and average method for updating the distance matrix.
- Using pairwise distances, the nearest neighbors are first combined into clusters, and then gradually linked together to form a single cluster.
- Single linkage method used the minimum distance between the pair points to update the distance matrix followed by maximum distance for complete linkage and average distance for average linkage.
- The authors claim that hierarchical algorithms reduce computational time significantly.
- The automated method for selecting the model parameters is one of the major highlights in the paper.

Overall the paper covers a wide range of methods with proper explanations and mathematical derivations which are useful for researchers and machine learning enthusiasts.

3.2.7 MacCrimmon (1968)

A survey on decision making techniques when there were several features to take into account was carried out by Kenneth R. MacCrimmon. He presents a novel strategy known as Multiple Criteria Decision Making (MCDM), which can help choose the best option for the given situation. The approach includes methods such as Utility Theory, Outranking Methods, Linear Programming, and WSM (Weighted Sum Method). This paper is very important to this dissertation since it selects the best model from all of the experiments.

- The Weighted Sum Method(WSM) is used to give a benchmark score for each alternative . It assign weights to each feature in the alternative and sums it up to give a single score to compare it with the other alternatives.
- The paper also explains about utility theory which frames a function according to the preferences.
- Linear Programming is an optimisation technique to determine the quantity needed to satisfy the goals or constraints of a project.
- Outranking Method is a pairwise comparison approach to seek out a better one between them.

Overall the authors work paved way for handling decision making systems and choosing the right alternative easy and still used for taking critical decisions.

3.3 Summary

- The dissertation identifies the model's limitations and analyzes the PhIDDLI pipeline with the assistance of Tsebriy et al. (2023).
- The limitations resulting from the large dimensionality of the dataset are confirmed by Tomašev & Radovanović (2016).
- Then, using Van Der Maaten & Hinton (2008) for t-SNE, McInnes et al. (2018) for UMAP, and The Greedy Choice (2017) for Isomap, the various dimensionality reduction strategies are investigated.
- For K-means clustering, MacQueen (1967) has been cited, while Murtagh & Contreras (2012) has been used for hierarchical clustering.
- In the meantime, all of the evaluation indicators have been given weights using MacCrimmon (1968), which is then combined to provide a single benchmark score for every model.

Chapter 4

Analysis and Diagnosis of PhIDDLI Pipeline

4.1 Analysis of the Pipeline

The following lists the stages of the PhIDDLI pipeline:

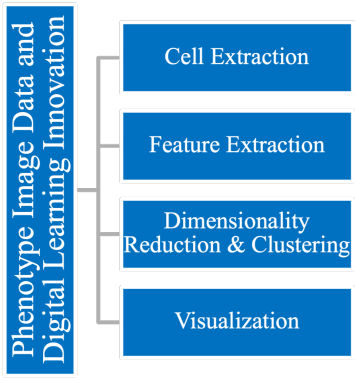


Figure 4.1: Stages of PhIDDLI pipeline(Tsebriy et al. 2023).

4.1.1 Cell Extraction

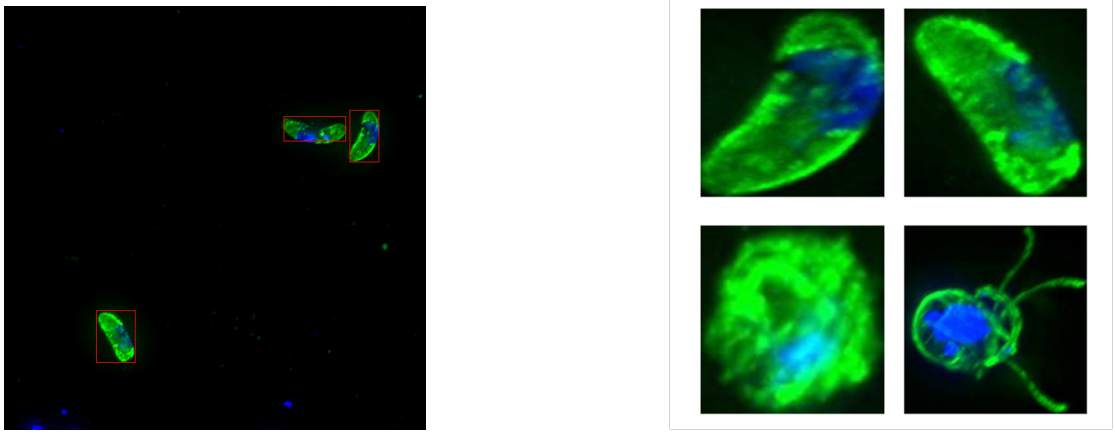
Currently, cell detection is carried out manually utilising the ICY open-source platform. The boundary boxes are documented in XML (Extensible Markup Language) format once the microscopic pictures are put into ICY and the cells are identified. The microscopic images are exported in TIF(Tagged Image Format) in order to prevent loss of quality. The exported TIF format images are converted in RGB(Red Green Blue) and saved as JPEG(Joint Photographic Experts Group) images. Moreover conversion of the XML documents to TXT(text) file types is also performed. The converted JPEG image and TXT file are used to extract the cells from the full image and store them as separate images. The script extracts the bounding box information from the TXT file and crops

the respective parts into individual images.

(a) Raw TIF microscopic image.

(b) RGB JPEG image.

Figure 4.2: Conversion of raw to RGB microscopic image.



(a) Cells detected using Bounding Boxes.

(b) Extracted cells from various images

Figure 4.3: Process involved in cell extraction.

4.1.2 Feature Extraction

To generate high-dimensional feature vectors, the extracted cells are now run through a pretrained model known as EfficientNet-B1. The extracted features contain 1000 dimensions. The extracted feature vectors can be used for clustering analysis and dimensionality reduction.

4.1.3 Dimensionality Reduction & Clustering

The high dimensional feature vectors are now modelled with K-means clustering. According to MacQueen (1967), initially centroids for each cluster has been assigned randomly. For each data point, the distance to each centroid is calculated using euclidean

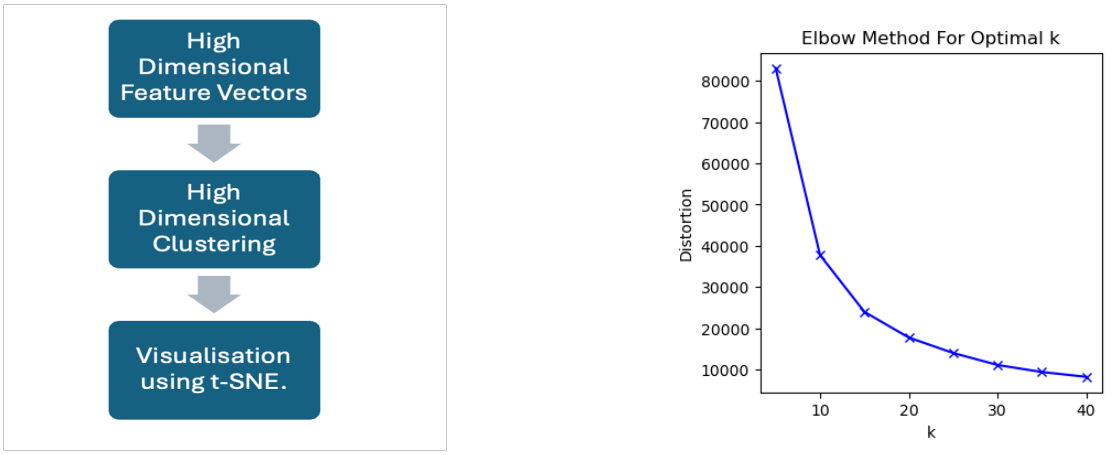
distance. The data point is assigned to the centroid which is closer than the other centroids. The equation for euclidean distance(MacQueen 1967) is given below:

$$C_i = \arg\min_j \|x_i - \mu_j\|^2 \tag{4.1}$$

where $\|x_i - \mu_j\|^2$ is the square of euclidean distance between data point x_i and centroid μ_j.
Then the mean value of the clusters are calculated by the following equation(MacQueen 1967).

$$\mu_j = \frac{1}{|C_j|} \sum_{x_i \in C_j} x_i \tag{4.2}$$

where C_j is the number of points in the cluster and $\sum_{x_i \in C_j} x_i$ is the sum of data points in the cluster. The new mean of the data points is considered the new centroid of the cluster, and the neighbour point assignment and mean calculation process continues until the cluster mean converges. The optimal number of clusters (k value) was determined by visualising the elbow curve, which is plotted between each k value and the distortion which is the sum of the squared distance between each data point and its assigned cluster. The optimal k value was determined using the KneeLocator Python tool. Currently, the values of k are being evaluated incrementally for each multiple of 5, such as 5, 10, 15, 20, 25, and so on. The clustered data points dimensions are reduced using t-SNE(t-distributed Stochastic Neighbour Embedding) for visualisation due to its non linear structure(Van Der Maaten & Hinton 2008).



(a) Current approach for clustering.

(b) Elbow curve for current approach.

Figure 4.4: The optimal number of cluster in this approach is found as 15.

4.1.4 Visualisation

The t-SNE tool initially calculates the pairwise distances(d_{ij}) for each set of points(Van Der Maaten & Hinton 2008). The distances are further calculated into conditional

probabilities. The equation for calculating conditional probabilities($p_{j|i}$) is given below:

$$p_{j|i} = \frac{\exp\left(-\frac{d_{ij}}{2\sigma_i^2}\right)}{\sum_{k\neq i}\exp\left(-\frac{d_{ik}}{2\sigma_i^2}\right)} \tag{4.3}$$

Here σ_i is the perplexity parameter that controls the width of the Gaussian kernel centered on point of interest(Van Der Maaten & Hinton 2008).The probability distribution is being reconstructed in lower dimensions using the student t-distribution. The euqation for calculating conditional probability in lower dimensions(q_{ij}) is :

$$q_{ij} = \frac{(1 + d_{ij}^{(\text{low})})^{-1}}{\sum_{k\neq l}(1 + d_{kl}^{(\text{low})})^{-1}} \tag{4.4}$$

Here $d_{ij}^{(\text{low})}$ is the pairwise distance in lower dimension and $\sum_{k\neq l}(1 + d_{kl}^{(\text{low})})^{-1}$ is the normalisation term so that the probabilities sum up to 1. Gradient descent algorithm(Nocedal & Wright 2006) is used to minimise the loss and KL divergence theorem(Van Der Maaten & Hinton 2008) is used to calculate the conversion loss.

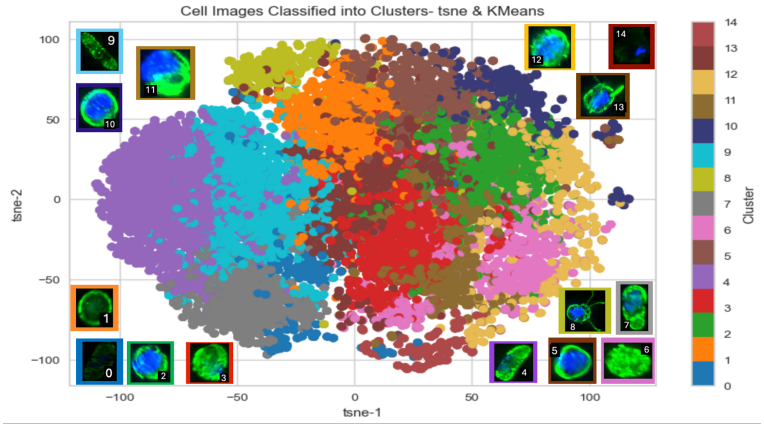


Figure 4.5: Visualisation of PhIDDLI model cluster using t-SNE(Tsebriy et al. 2023) with a sample image from each cluster.

4.1.5 Early analysis of cluster images

According to Tsebriy et al. (2023), clusters 4,9 and 7 represented coffee bean shaped gametocytes which indicates that they have been **arrested early** by drugs. Furthermore, clusters 1,3 and 6 have rounded gametocytes with minimal or no blue patches, indicating that the gametocytes were **arrested in the middle stage**. Clusters 2,5,10,11,12 and 13 had rounded cells with bright blue areas, indicating gametogenesis, but they were arrested before fully exflagellating. Cluster 8 had fully exflagellated cells with tentacles indicating that these cells are **arrested very late**. Clusters 0 and 14 represented

unclear images. Furthermore, the cluster points appear with extensive overlaps and confusing boundaries.

4.2 Diagnosis of PhIDDLI model

4.2.1 Average Majority Share Calculator

A customised CNN(Convolutional Neural Network)(LeCun et al. 1998) model has been used to verify how well the images fit each cluster using the acquired domain knowledge. A subset of labelled gametocyte images are collected to train the model. The following are the mainly categorised five labels for the training:

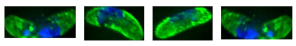
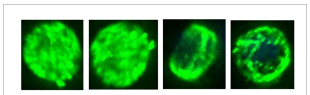
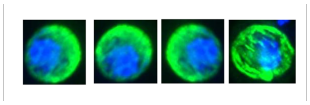
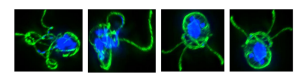
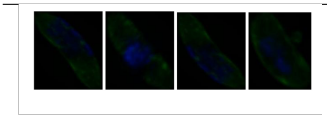
Label Image	Label Explanation
	These images describe that the gametocytes have been arrested early.
	These images describe that the gametocytes have been arrested mid level 1. It is ensured that the images have either extremely few or no blue patches.
	These images describe that the gametocytes have been arrested mid level 2. It is ensured that the images have bright blue patches.
	The tentacles showed that the cells have completed gametogeneis. These cells can be termed as arrested late.
	Unclear images are labelled as noise .

Table 4.1: Table displaying the labels and sample images from the training dataset.

The architecture contains four convolutional layers(LeCun et al. 1998) for extracting the features and two maxpool layers to downsample the feature map. The downsampled feature map is then processed by four fully connected layers with 512, 256, 128 and 30 neurons respectively. ReLU(Rectified Linear Unit) has been applied for each convolutional and fully connected layer to provide non linear structure. Before processing the images, data augmentation techniques such as horizontal flipping and resizing to 64x64 were used. Cross entropy loss(Goodfellow et al. 2016) is used to calculate the loss in each epoch and Adam(ADAptive Moment estimator)(Kingma & Ba 2014) which is an extensive version of stochastic gradient descent has been used for optimisation. The network was trained for 15 epochs.

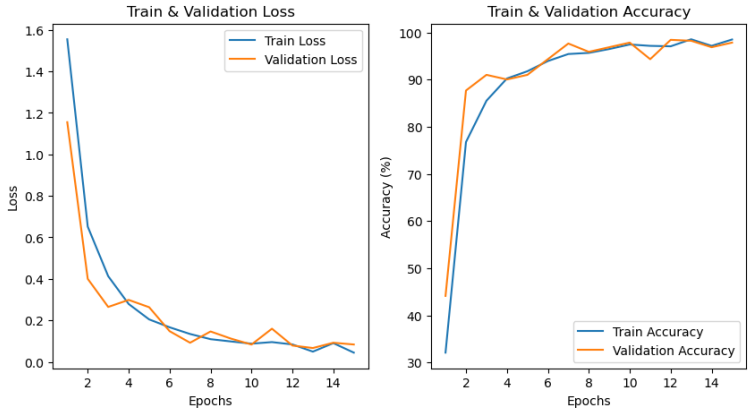


Figure 4.6: The dataset is split into training and validation data and the loss and accuracy is plotted .

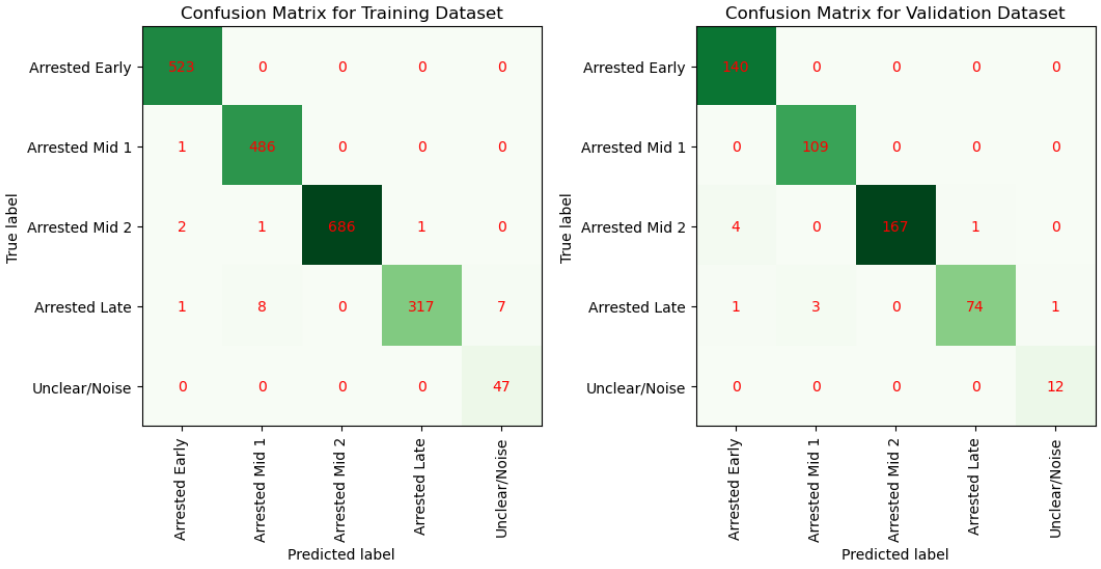


Figure 4.7: Confusion matrices for training and validation dataset.

Using this model, the class which holds the majority share in a cluster is decided as the label for the cluster. For example from **Figure 4.8**, the majority share in cluster 4 is 98.07% which belongs to "arrested early" cells. So the label for cluster 4 can be labelled as cluster with "arrested early" cells. The model's average majority share is calculated by taking the mean of the majority share from all the clusters. In PhIDDLI model, the average majority share is 65.9%. This means on an average each cluster in the PhIDDLI model contains 65.9% images from the same class.

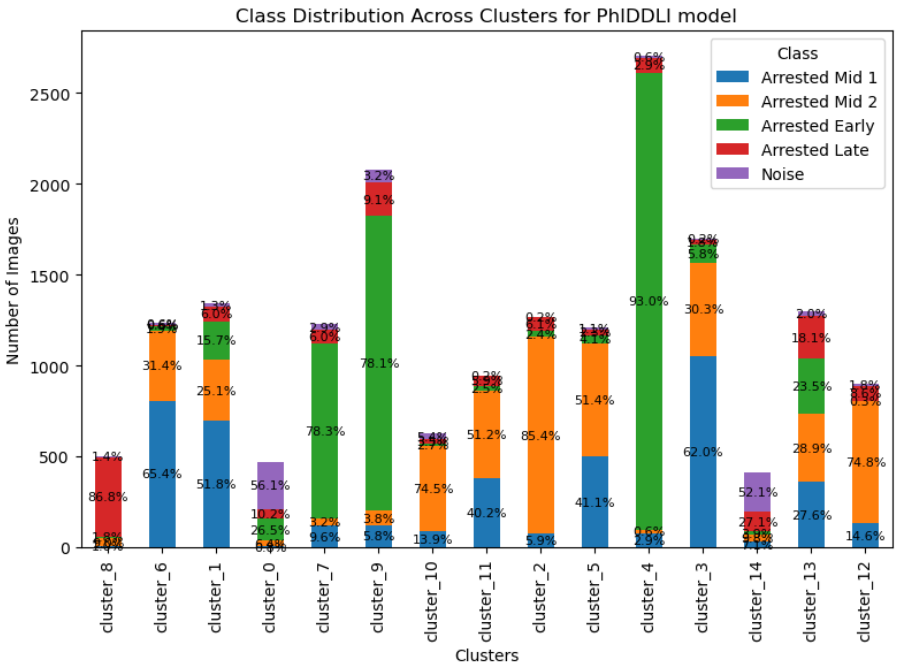


Figure 4.8: Majority Share in each cluster for PhIDDLI model.

Although the 'arrested mid level 2' class has the largest share in cluster 13, at 28.9%, no class has a single majority at more than 50%, indicating that the cluster contains mixed images that may not be helpful for interpretation. Nonetheless, the model can efficiently separate the "arrested early" class and can also decently segregate other classes. Additionally, the overlapping of borders from **Figure 4.5** makes it difficult to understand images in those regions.

4.2.2 Silhouette Score

Silhouette score(Rousseeuw 1987) is a metric to determine the quality of clustering in the model. It indicates how similar a data point to its own cluster. The equation to calculate the silhouette score s(i) is:

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))} \tag{4.5}$$

where,

- a(i) is the average distance between point i and rest of the points in the same cluster,
- b(i) is the average distance between point i to other points in the nearest other cluster.

The score value ranges from -1 to +1 and higher values indicate good clustering. The

sklearn.metrics package is used to determine the silhouette score for this model. The result achieved is 0.0891 which very low for a clustering model.

4.2.3 Calinski-Harabasz Index

The Calinski Harabasz index(Calinski & Harabasz 1974) determines how well the clusters are separated.The equation to calculate the Calinski-Harabasz index CH is:

$$CH = \frac{\text{tr}(B_k)}{\text{tr}(W_k)} \cdot \frac{n-k}{k-1} \tag{4.6}$$

Where:

- $\text{tr}(B_k)$: The sum of squared deviations between the cluster centers and the overall mean of the data.(inter cluster dispersion)
- $\text{tr}(W_k)$: The sum of squared deviations of each data point from its corresponding cluster center.(intra cluster dispersion)
- n: Number of data points.
- k: Number of clusters.

The score value ranges from 0 to ∞ and higher values indicate good clustering. Using sklearn.metrics tool, the result achieved is 4935.98. This score can be compared with other experiments in order to determine the model's performance.

4.2.4 Davies Bouldin Index

The Davies Bouldin index(Davie s & Bouldin 1979) indicates the compactness of the clusters. The equation to calculate the silhouette score DB is:

$$DB = \frac{1}{k} \sum_{i=1}^{k} \max_{j \neq i} \left(\frac{S_i + S_j}{M_{ij}} \right) \tag{4.7}$$

where:

- k is the number of clusters.
- S_i is the average distance between each point in cluster i and the centroid of cluster i.
- M_{ij} is the distance between the centroids of clusters i and j.
- $\max_{j \neq i} \left(\frac{S_i + S_j}{M_{ij}} \right)$ represents the maximum ratio for each cluster i with respect to all other clusters j.

The score ranges from 0 to ∞ and lower values indicate good compactness of the clusters resulting in a good model.Using sklearn.metrics tool, the result achieved is 1.809.

Chapter 5

The Curse of Dimensionality

5.1 Data Lost in space

As the dimensions grow, there's a potential that the feature points will get buried in the high dimensional space, making the data points sparser and more challenging to interpret(Altman & Krzywinski 2018). Three dimensional data points from a random hundred images are taken from the primary screen image embedding dataset. At first, the data points are plotted with only one of the three dimensions, then two of the three dimensions, and finally all three. The one-dimensional plot is now divided into equal-spacing lines, followed by equal-spacing squares for the two-dimensional data points and equal-sized cubes for the three-dimensional data points.

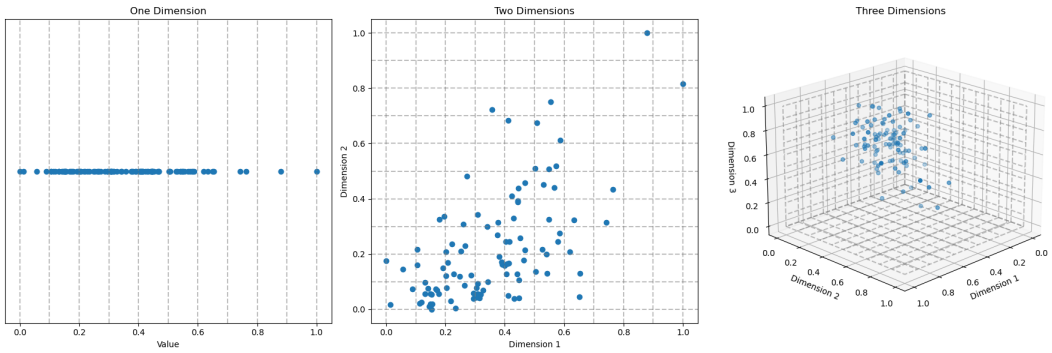


Figure 5.1: A visual representation on how data points occupy space in different dimensions.

Plots like the ones above show that as dimensions rise, there are more empty cells.Nearly every one of the ten cells in a one-dimensional plot has at least one point in it. There are 64 empty squares and 928 empty cubes without a single data point in the two-dimensional plot and three-dimensional plot, respectively.Given that the dataset contains 1000 dimensions, there is a good chance that the points will become increasingly dispersed, giving the impression that the data is buried in higher dimensions and

more difficult to interpret.

5.2 Hubness

Hubness is a concept that occurs in high-dimensional data, where some points (or data items) become "hubs" or central points that are very frequently the nearest neighbours to other points(Tomašev & Radovanović 2016). These hubs make the distribution of neighbour occurrence frequency skewed. Neighbour occurrence frequency is a measure used to quantify how often a particular data point appears as a nearest neighbour to other points in a dataset. This can cause problems in data analysis because these hubs can distort the results of clustering algorithms, making them less accurate.
In order to examine the hubness impact in the primary screen image embedding dataset, a random dimension is initially selected from a collection of 1000 dimensions. For the neighboring occurrence frequencies, a probability density curve is plotted. For 2,100 and 1000 dimensions, the same process is done.

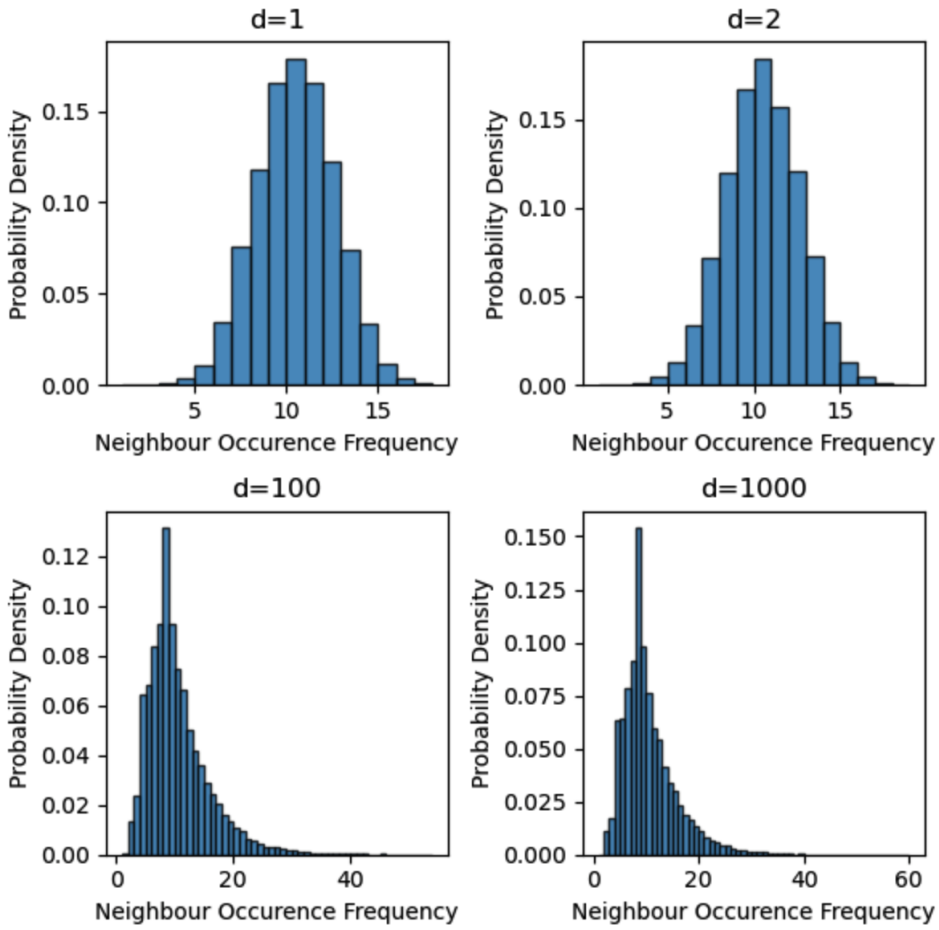


Figure 5.2: Distributions of Neighbour occurence frequency at 1,2,100 and 1000 dimensions for Primary Screen Dataset.

The formula for calculating skewness of the neighbour occurence($SN_k(x)$) frequency is given below(Tomašev & Radovanović 2016):

$$SN_k(x) = \frac{m_3(N_k(x))}{m_2^{3/2}(N_k(x))} = \frac{\left(\frac{1}{N}\sum_{i=1}^{N}(N_k(x_i) - k)^3\right)}{\left(\frac{1}{N}\sum_{i=1}^{N}(N_k(x_i) - k)^2\right)^{3/2}} \tag{5.1}$$

Where:

- $N_k(x)$represents the neighbor occurrence frequency.
- $m_3(N_k(x))$ is the third central moment of the neighbour occurence frequency distribution of $N_k(x)$ around the mean, capturing the skewness.
- $m_2(N_k(x))$ is the second central moment (variance) of the neighbour occurence frequency distribution.

From the above plot we can observe that while dimension increases in this dataset, the skewness in the distribution increases too indicates the presence of hubness in this dataset. The skewness of these distributions are calculated as -0.009939694892076769, -0.019795075849741002, 1.6608874957623627, 1.6948153769840806 for 1,2,100 and 1000 dimensions respectively. The skewness has been calculated using skew package in python.

5.3 Instability of Evaluation Metrics

Evaluation metrics with two random dimensions are initially computed for the main screen image embedding dataset, including silhouette score, Calinski-harabaz, and Davies Bouldin indices. Next, 100 random dimensions are calculated, then 200, 300,... 1000 dimensions(Tomašev & Radovanović 2016). Plotting the scores and examining the trend are now being performed.

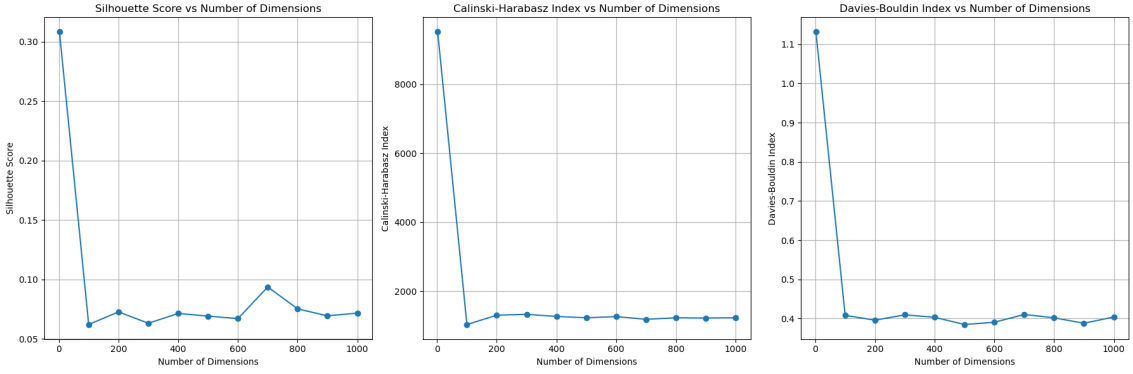


Figure 5.3: Dimensions vs Evaluation Metrics.

The greater values in the Davies-Bouldin index denote worse quality, so the values are inverse. As dimensions increase, it is now recognized that all three measurements show a considerable decline in quality indices, suggesting issues with high-dimensional space modeling.(Tomašev & Radovanović 2016)

5.4 Observations so far

- The dataset suffers from the curse of dimensionality, as evidenced by the dimensionality tests in Chapter 5, with 1000 dimensions of embeddings produced by Efficient-Net B1.
- Low dimensionality modeling of the dataset can be used to examine how well the model fits the data points which will be implemented in Chapter 6.
- Regarding dimensionality reduction, several experiments can be carried out. In contrast to t-SNE, which arrived in 2008, Isomap and UMAP arrived in 2017 and 2018, respectively, making them the most recent arrivals for the experiments.
- Regarding clustering techniques, several experiments can be carried out. Clustering experiments using K-means and hierarchical experiments will take place under each dimensionality reduction.
- In K means clustering, multiples of 1 can be examined rather than k values with multiples of 5.
- The borders between the clusters are unclear and frequently overlapping. The goal of the experiment is to address these issues as much as possible.
- Additionally, because the boundaries overlap too much, it is observed that the PhIDDLI model is not appropriate for identifying hidden patterns between the boundaries.
- The chosen model from the multi attribute method is then transformed to a fuzzy clustering model by observing at the soft clustering models from Xu & Wunsch (2010) in order to find deep mixed patterns irrespective of the four primary classes.

Chapter 6

Experiments using Isomap and UMAP Data Points

6.1 Explanation of Isomap with example

Isomap(Isometric Mapping) reduces the dimension by making neighbourhood graphs with the help of nearest neighbours(The Greedy Choice 2017) and calculate the similarity scores using Floyd-Warshall Algorithm(Risald & Suyoto 2017). Then the Multi Dimensional Scaling(MDS)(Carroll & Arabie 1998) is used to reconstruct the data points in low dimensional space. To study and observe the algorithm, a random small subset of data points with two random dimensions from the image embedding dataset is selected to undergo the process. The sample is shown below:

Point	Coordinates
Point 1	$[-0.70323211, 0.89485216]$
Point 2	$[-0.85881805, 0.4431116]$
Point 3	$[-1.07164979, 1.42083216]$
Point 4	$[-1.52713239, 1.84265852]$

Table 6.1: Coordinates of Points

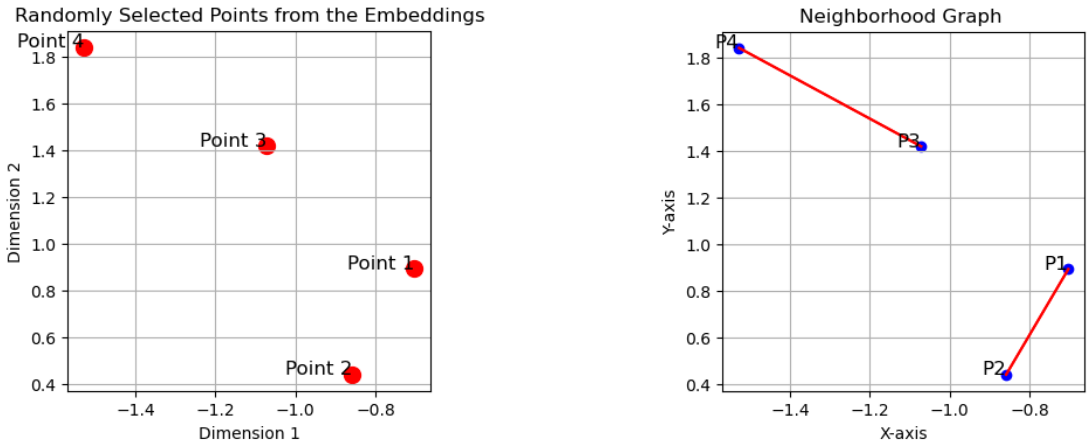
6.1.1 Step 1: Finding the nearest neighbours

The pairwise distance for each pair of points is calculated using euclidean method and the nearest neighbours are chosen. The pairwise distances are calculated and the table is provided below.

	Point 1	Point 2	Point 3	Point 4
Point 1	0	0.47766	0.64208	1.25665
Point 2	0.47766	0	1.00076	1.55113
Point 3	0.64208	1.00076	0	0.62155
Point 4	1.25665	1.55113	0.62155	0

Table 6.2: Distance Matrix for pairwise distances (Euclidean).The nearest distances are highlighted in green.

For ease of comprehension, the number of closest neighbours in this case is two (k=2). Counting the nearest neighbour includes the point of interest too. Now the neighbourhood graph is plotted with the help of nearest neighbours.



(a) Sample Dataset.

(b) Neighbourhood Graph.

Figure 6.1: Plotting the Neighbourhood Graph.

6.1.2 Step 2: Finding the shortest path

Next step involves finding the geodesic distance which is the shortest path between all pair of points and updating the distance matrix(The Greedy Choice 2017) as it makes the method non linear. Floyd-Warshall algorithm is used here to find the shortest path as the points lie on a graphical plane. The equation for the corresponding algorithm is given below:

$$\text{dist}(i,j) = \min(\text{dist}(i,j), \text{dist}(i,k) + \text{dist}(k,j)) \tag{6.1}$$

where,

- i,j are the data points,
- k is the intermediate point.

At first, a particular point is assigned as the intermediate point(k), and the remaining points are assigned as i and j to perform the iterations. Here at first point 1 is considered

as intermediate, then the equation would look like:

$$\text{dist}(i, j) = \min(\text{dist}(i, j), \text{dist}(i, 1) + \text{dist}(1, j)) \tag{6.2}$$

If there is no connection between two points, the corresponding cell in the distance matrix is set to infinity. Now the initial distance matrix will be:

	Point 1	Point 2	Point 3	Point 4
Point 1	0	0.47766	∞	∞
Point 2	0.47766	0	∞	∞
Point 3	∞	∞	0	0.62155
Point 4	∞	∞	0.62155	0

Table 6.3: Initial Distance matrix

Now points 2 and 3 are considered i and j and the calculation is shown below:

$$\text{dist}(2, 3) = \min\left(\text{dist}(2, 3),\ \text{dist}(2, 1) + \text{dist}(1, 3)\right)$$

$$\text{dist}(2, 3) = \min\left(\infty,\ 0.47766 + \infty\right) = \infty$$

$$\text{dist}(2, 3) = \min(\infty, 0.47766 + \infty) = \infty$$

Further the iteration is repeated for points 3 and 4 as i and j and 2 and 4 as i and j to complete the distance matrix for point 1 as intermediate node. The iteration is mentioned as pseudocode(Wikipedia 2024) below:

Algorithm 1 Floyd-Warshall Algorithm

```
Let dist be a |V| × |V| array of minimum distances initialized to ∞ (infinity)
for each edge (u, v) do
    dist[u][v] ← w(u, v) {The weight of the edge (u, v)}
end for
for each vertex v do
    dist[v][v] ← 0
end for
for k from 1 to |V| do
    for i from 1 to |V| do
        for j from 1 to |V| do
            if dist[i][j] > dist[i][k] + dist[k][j] then
                dist[i][j] ← dist[i][k] + dist[k][j]
            end if
        end for
    end for
end for
```

Now the iteration is repeated by assigning points 2,3 and 4 as intermediate point and the distance matrix with smallest values are finalised . In this case, all the intermediate

points gave the same distance matrix so the initial matrix will be the final distance matrix.

6.1.3 Step 3: Multi Dimensional Scaling(MDS)

The final step involves eigen decomposition of the distance matrix and then the largest eigenvalue and its corresponding eigenvector is chosen to calculate the low dimensional data points(Abdi & Williams 2010). For practical purposes, the infinity is replaced by arbitrarily large values.The steps for MDS (The Greedy Choice 2017) is mentioned below: **Step-by-Step Process:**

Equation for the centered distance matrix B:

$$B = -\frac{1}{2} H D^2 H \tag{6.3}$$

where D^2 is the matrix of squared distances and H is the centering matrix.The formula for centering matrix(H) is :

$$H = I - \frac{1}{n}\mathbf{1}\mathbf{1}^T \tag{6.4}$$

Where:

I is the identity matrix of size $n \times n$,

n is the number of points,

$\mathbf{1}$ is a column vector of ones of size n.

The value of n is assigned as 4 as there are four data points in the sample and the matrix of ones($\mathbf{1}\mathbf{1}^T$) is given as

$$\mathbf{1}\mathbf{1}^T = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \tag{6.5}$$

Therefore by substituting the values in (4), the centering matrix is obtained as

$$H = I - \frac{1}{n}\mathbf{1}\mathbf{1}^T = \begin{pmatrix} 0.75 & -0.25 & -0.25 & -0.25 \\ -0.25 & 0.75 & -0.25 & -0.25 \\ -0.25 & -0.25 & 0.75 & -0.25 \\ -0.25 & -0.25 & -0.25 & 0.75 \end{pmatrix} \tag{6.6}$$

Now the centering matrix(H) and the distance matrix(D) can be used to calculate the

centered distance matrix(B):

$$B = \begin{pmatrix} -0.125388 & 0.125388 & -24.76043 & 24.76043 \\ 0.125388 & -0.125388 & 24.76043 & -24.76043 \\ -24.76043 & 24.76043 & -0.125388 & 0.125388 \\ 24.76043 & -24.76043 & 0.125388 & -0.125388 \end{pmatrix} \tag{6.7}$$

Then the centered distance matrix(B) is undergone for eigen decomposition

$$B = V \Lambda V^T \tag{6.8}$$

The largest eigenvalue λ_1 and its corresponding eigenvector v_1 is used to find the 1D embeddings.

$$Y = \sqrt{\lambda_1} \cdot v_1 \tag{6.9}$$

Using Python for eigen decomposition, the following 1D coordinates[Y] are obtained:

$$Y = \begin{bmatrix} -0.5 \\ 0.5 \\ -0.5 \\ 0.5 \end{bmatrix}$$

6.1.4 Hyperparamter Tuning

The Isomap is applied on the high dimensional image embedding dataset. The following are the algorithm's hyperparameters:

- **n-neighbours:** Number of neighbours to be considered by the point of interest.
- **n-components:** Number of dimensions which is set to 2 for experimenting in low dimensions.

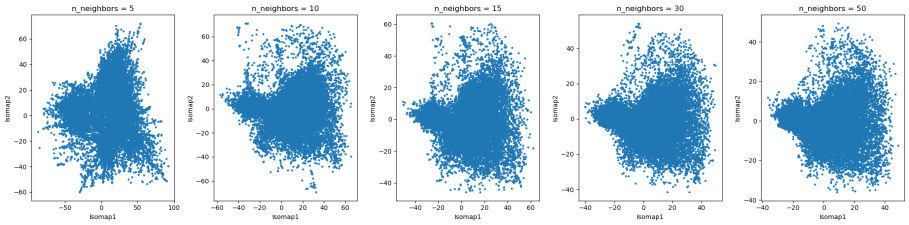
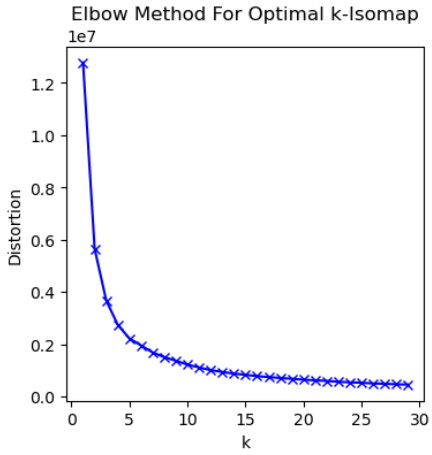


Figure 6.2: Isomap technique applied with varying n-neighbours.

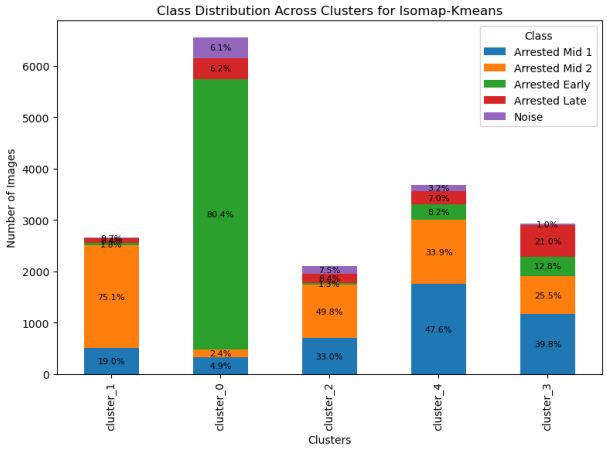
Despite the similar appearance of all the versions, denser zones are found where n-neighbors=15 and 30, and clustering is carried out with n-neighbors=15.

6.1.5 Experiment 1: Clustering using Kmeans on Isomap data

Using the KneeLocator Python module, the ideal number of clusters is discovered to be five after subjecting the low dimensional data points to k means clustering. The elbow curve is plotted for multiples of 1 as k values.



(a) Elbow curve for Isomap-Kmeans model.



(b) Purity of each cluster in the Isomap-Kmeans model.

Figure 6.3: Isomap-Kmeans model elbow curve and Majority Share in each cluster.

In **Figure 6.3(b)** a large portion of the images from the clusters 0 and 1 belongs to the classes "arrested early" and "arrested mid 2" respectively. Almost half of the images from cluster 2 belongs to the "arrested mid 2 " class. Despite the fact that the "arrested mid 1" class comprises the majority of the photos in clusters 3 and 4, the class's share of images is less than 50% in each cluster. It is noted that "arrested late" didn't get majority in any of these clusters. The average majority share in each cluster is 58.4%.

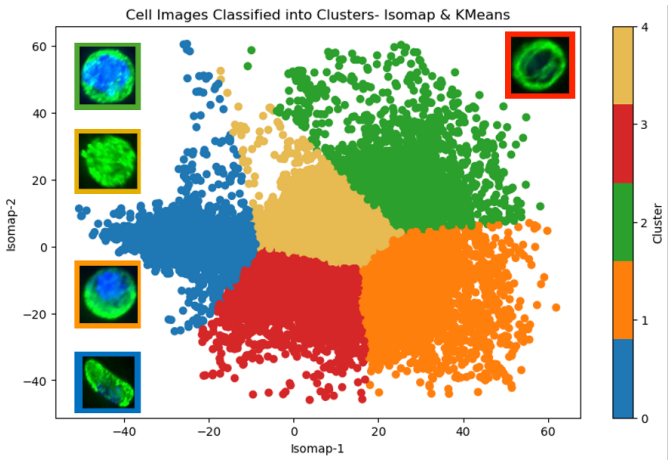


Figure 6.4: Isomap-Kmeans Clustering model with sample cluster images.

It can be seen from **Figure 6.4** that the clusters appear to be closer to one another,

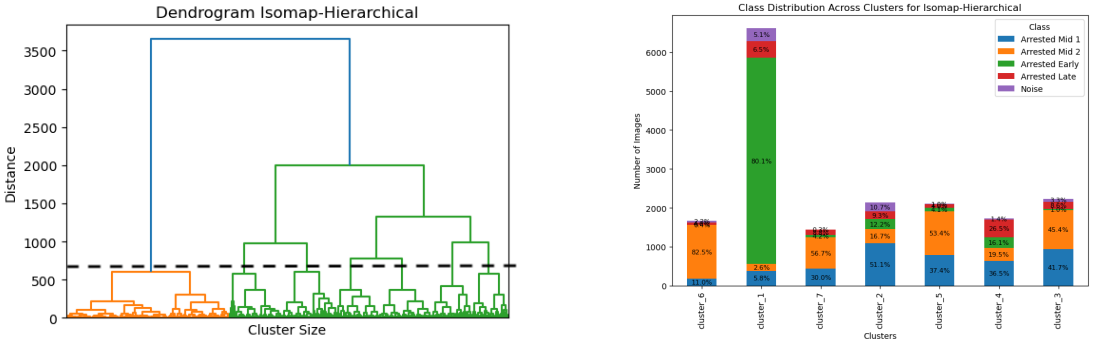
overlapping at the edges. This could be the cause of the subpar results in clusters 2, 3, and 4. Although cluster 3's performance was impacted by sharing boundaries with clusters 2 and 4, it should have ideally had a majority of "arrested late" classes. The following are the model's evaluation metrics:

Metric	Score
Average Majority Share	58.40%
Silhouette Score:	0.47380478
Calinski-Harabasz Index	21305.6286
Davies-Bouldin Index:	0.83262452

Table 6.4: Clustering Evaluation Metrics for Isomap-Kmeans model.

6.1.6 Experiment 2: Hierarchical Clustering on Isomap Data

Plotting the dendrogram(refer **Figure 6.5(a)**), the cut for the ideal number of clusters was done at a distance of 530 . Given that the cut in **Figure 6.5(a)** travels through seven lines, seven is decided as the number of perfect clusters.



(a) Dendrogram for Isomap-Hierarchical model.

(b) Purity of each cluster in the Isomap-Hierarchical model.

Figure 6.5: Isomap-Hierarchical model Dendrogram and Majority Share in each cluster.

It can be seen from **Figure 6.5(b)** that clusters 1, 2, 5, 6, and 7 can have a majority from a single class that is exceeding 50%. However clusters 3 and 4 does not hold a single majority. Moreover none of the clusters had "arrested late" as the majority class. The clustering technique is able to separate the gametocytes shaped like coffee beans from the other photos very successfully, as seen by the fact that 80.1% of the images in cluster 1 belong to the "arrested early" class. The average majority share in each cluster is 57.9%.

Figure 6.7 depicts a scenario similar to the K-means model, where all the clusters exhibit overlapping and close proximity to one another.

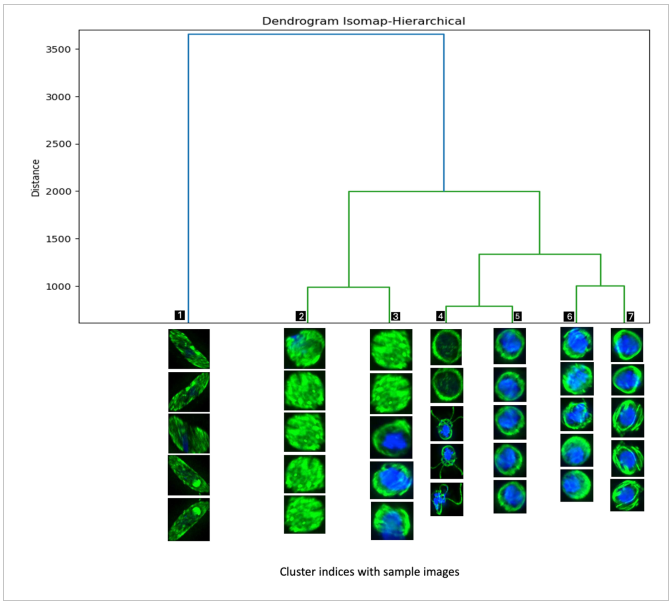


Figure 6.6: Isomap-Hierarchical Dendrogram with sample cluster images.

The borders appear to be more non-linear, contrasting with the linear boundaries observed in the k-means model.Clusters 5,6 and 7 have "arrested mid 2" as the majority class. However slight changes are observed among these clusters. While cluster 5 exhibited a fairly spherical structured cells, cluster 6 displayed cells with erratic spherical patterns with a blue nucleus. In this case, images in cluster 6 were very close to being exflagellated. The evaluation metrics for this model are provided in **Table 6.5**.

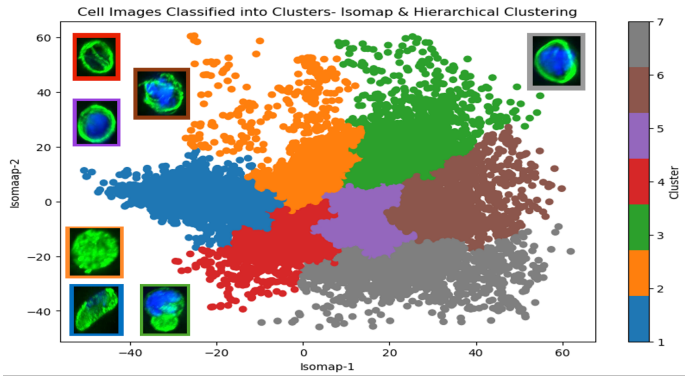


Figure 6.7: Isomap-Hierarchical clustering model with sample cluster images.

Evaluation Metrics	**Score**
Average Majority Score	57.9%
Silhouette Score	0.4121
Calinski-Harabasz Index	16593.1718
Davies-Bouldin Index	0.9105

Table 6.5: Clustering Evaluation Metrics for Isomap-Hierarchical model.

6.2 Explanation of UMAP with example

After determining pairwise distances, UMAP (Uniform Manifold Approximation and Projection)(McInnes et al. 2018) employs an exponential distribution to determine the similarity scores. Following the computation of similarity scores, the student t-distribution is used to reconstruct the data points in lower dimensions in a manner that's similar to the high dimensional structure(McInnes et al. 2018). For reconstruction, the cross entropy loss function is utilised, and for optimisation, stochastic gradient descent is employed. Whereas in t-SNE, gaussian kernel is used to calculate the similarity score. Here, the same set of data points used for Isomap illustration have been used.

6.2.1 Step 1: Finding the pairwise distances and nearest neighbours

The pairwise distances and nearest neighbors are the same as in **Table 6.2** since the same data points used for Isomap example are utilized here.

6.2.2 Step 2: Computing the similarity score

UMAP calculates the similarity score(μ_{ij}) by using the equation below:

$$\mu_{ij} = \exp\left(\frac{-\max(0, d_{ij} - \rho_i)}{\sigma_i}\right) \tag{6.10}$$

where,

- μ_{ij} is the similarity score.
- d_{ij} is the distance between point x_i and x_j.
- ρ_i is the nearest neighbour.
- σ_i is the local scale parameter which is $\log_2(k)$.In this case k=2, therefore the local scale parameter is 1.

A sample calculation for similarity score between point 1 and point 2 is shown below:

$$\mu_{12} = \exp\left(\frac{-\max\left(0, 0.4778 - 0.4778\right)}{1.0}\right) = \exp(0) = 1$$

Likewise the similarity score for all pairwise distances are calculated and the results are tabulated below:

	Point 1	Point 2	Point 3	Point 4
Point 1	1.6124954	1.0	0.84841075	0.45929488
Point 2	1.0	1.6124954	0.59283788	0.34193165
Point 3	0.97886077	0.6839915	1.86042962	1.0
Point 4	0.52991518	0.3945064	1.0	1.86042962

Table 6.6: UMAP Similarity Scores (μ)

6.2.3 Step 3: Computing the symmetrical similarity score

The formula to symmetrize the similarity score(w) is given below:

$$w_{ij} = \mu_{ij} + \mu_{ji} - \mu_{ij} \cdot \mu_{ji} \tag{6.11}$$

A sample calculation between point 1 and point 3 is stated below:

$$w_{13} = \mu_{13} + \mu_{31} - \mu_{13} \cdot \mu_{31} = 0.8484 + 1.0 - (0.8484 \times 1.0) = 0.99679552$$

Likewise the symmetrical similarity score for all the values are calculated and tabulated below:

	Point 1	Point 2	Point 3	Point 4
Point 1	0.62484938	1.0	0.99679552	0.74582273
Point 2	1.0	0.62484938	0.87133331	0.60154383
Point 3	0.99679552	0.87133331	0.25966087	1.0
Point 4	0.74582273	0.60154383	1.0	0.25966087

Table 6.7: Symmetrical Similarity Score (w)

6.2.4 Step 4: Reconstruction in Lower Dimension

Now randomly data points are initialised in lower dimensions and iteratively gets updated to minimise the loss function. Cross entropy function is used to calculate the loss and stochastic gradient descent has been used for optimisation(McInnes et al. 2018). The symmetric similarity scores are converted into probabilities(p_{ij}) using(McInnes et al. 2018):

$$p_{ij} = \frac{w_{ij}}{\sum_{k \neq l} w_{kl}} \tag{6.12}$$

where,

- p_{ij} is the probability score between points i and j.
- w_{ij} is the symmetrical similarity score between points i and j.
- k and l are random data points.

- $\sum_{k \neq l} w_{kl}$ is the sum of all pairwise symmetrical similarity scores under one condition that k and l are not the same data point.

Now the obtained values are used to calculate the cross entropy loss(C) by(McInnes et al. 2018):

$$C = \sum_{i \neq j} \left[p_{ij} \log \left(\frac{q_{ij}}{p_{ij}} \right) + (1 - p_{ij}) \log \left(\frac{1 - q_{ij}}{1 - p_{ij}} \right) \right] \tag{6.13}$$

where,

- p_{ij} is the probability scores for high dimensional pairwise data points.
- q_{ij} is the probability scores for low dimensional pairwise data points.

After sufficient iterations, the algorithm converges with a configuration where the local and global structure of the high dimensional dataset is retained in low dimension. The algorithm is done using UMAP python tool and the low dimensional data points[Y] are obtained below:

$$Y = \begin{bmatrix} 32.97621 \\ 32.166805 \\ 30.985752 \\ 30.07776 \end{bmatrix}$$

The key differences between t-SNE and UMAP are listed below:

Feature	**t-SNE**	**UMAP**
Similarity Measure in High Dimensions	Gaussian Kernel	Exponential Distribution
Loss Function	KL Divergence	Cross Entropy Loss

Table 6.8: Comparison between t-SNE and UMAP.

6.2.5 Hyperparamter Tuning

The UMAP algorithm is applied on the high dimensional image embedding dataset. The following are the hyperparameters for the algorithm.

- **n-neighbours:** Number of neighbours to be considered by the point of interest.
- **n-components:** Number of dimensions which is set to 2 for experimenting in low dimensions.
- **min-dist:** The minimum distance to keep between points in lower dimensional space.

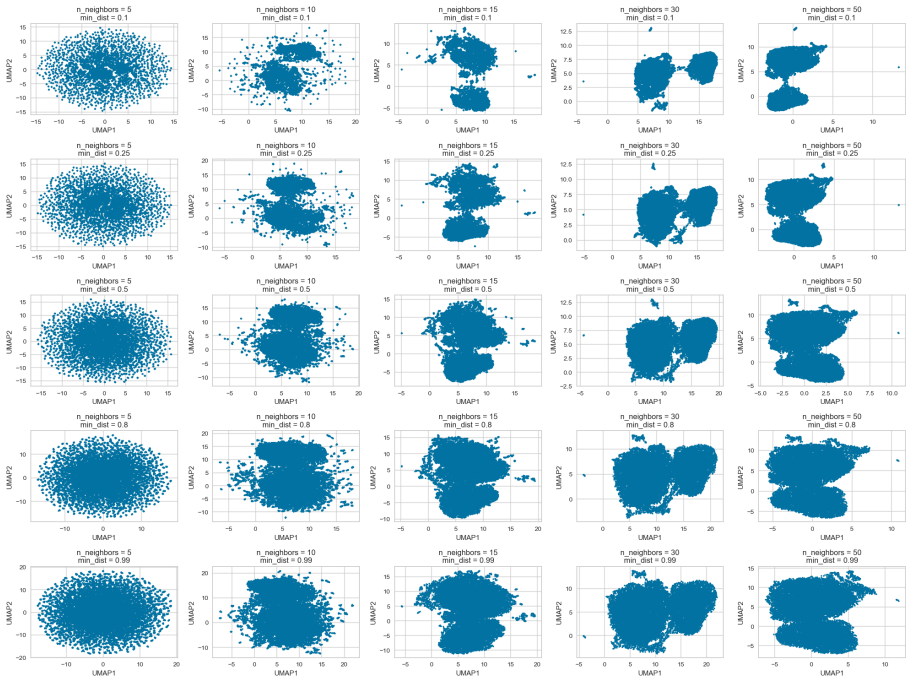
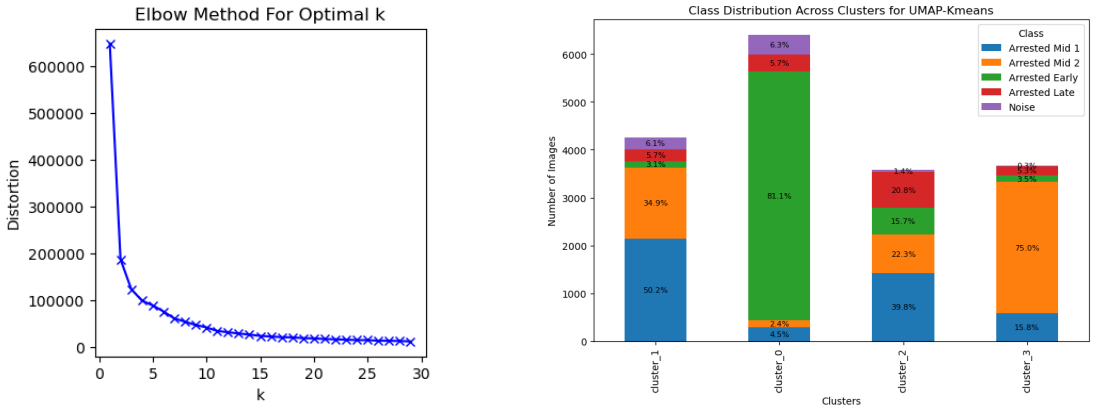


Figure 6.8: UMAP structures with varying hyperparameters.

The UMAP structure is finalised at n-neighbours=15 and min-distance=0.1 as the clusters look quite well separated than other versions.

6.2.6 Experiment 3: Kmeans Clustering on UMAP data

Similar to experiment 1, KneeLocator Python module has predicted the ideal number of clusters as four after modeling on the UMAP structure with multiples of 1 as k values.



(a) Elbow curve for UMAP-Kmeans model.

(b) Purity of each cluster in the UMAP-Kmeans model.

Figure 6.9: UMAP-Kmeans model elbow curve and Majority Share in each cluster.

Here , cluster 0 has cells belonging to arrested early class with 81.1% majority share.

Moreover clusters 1 and 2 has majority of the cells belonging to arrested mid 1 class with 50.2% and 39.8% majority shares respectively. Also it is noted that most of the arrested late cells belongs in cluster 2 even though mid 1 is the majority in that cluster. Cluster 3 has "arrested mid level 2" cells with 75% majority share. The average majority share for this model is 61.5%.

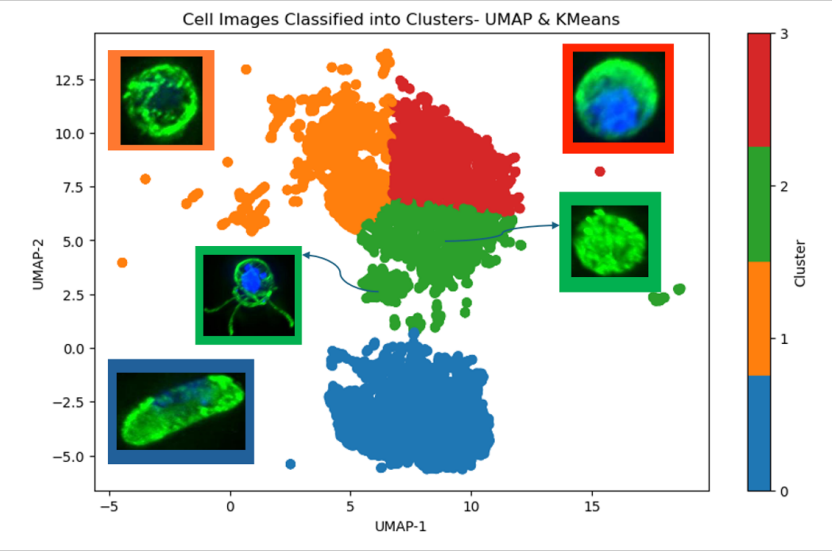


Figure 6.10: UMAP-Kmeans model with a sample image from each cluster.

UMAP has successfully distinguished the arrested early coffee bean-shaped cells from the remaining clusters, in contrast to Isomap models. The remaining clusters, however, struggle with boundary sticking. Arrested late cells have developed into a separate cluster in cluster 2, situated between cluster 0 and the other clusters. However algorithm included it in the cluster 2.

Evaluation Metrics	**Score**
Average Majority Score	61.5%
Silhouette Score	0.4642558
Calinski-Harabasz Index	33310.1523
Davies-Bouldin Index	0.83459

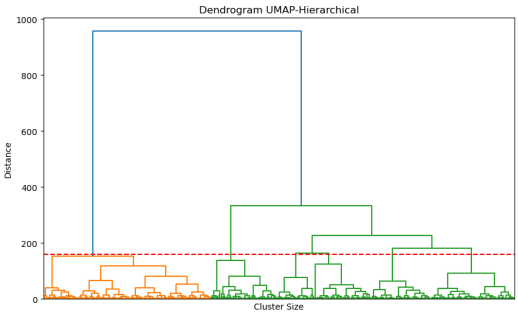
Table 6.9: Clustering Evaluation Metrics for UMAP-Kmeans model.

6.2.7 Experiment 4: Hierarchical Clustering on UMAP data

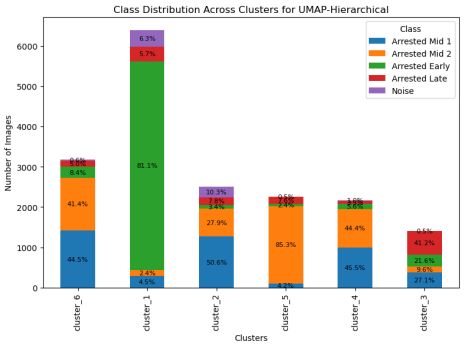
While plotting the dendrogram(refer **Figure 6.11(a)**), the cut for the optimal number of clusters was made at a distance of 160. Six are the number of ideal clusters, since the cut in **Figure 6.11(a)** passes across seven locations.

Cluster 1 has a majority share of early coffee bean-shaped gametocytes that have been arrested, which is similar to the Kmeans model. In cluster 2, "Arrested mid level

2" cells hold a majority share of 50.6%. Arrested mid level 1 and 2 share approximately equal percentages in clusters 4 and 6. In cluster 3, the majority of arrested late cells are 41.2%. This model experiences the same sticking boundary issue as previous experiments.



(a) Elbow curve for UMAP-Hierarchical model.



(b) Purity of each cluster in the UMAP-Hierarchical model.

Figure 6.11: UMAP-Hierarchical model elbow curve and Majority Share in each cluster.

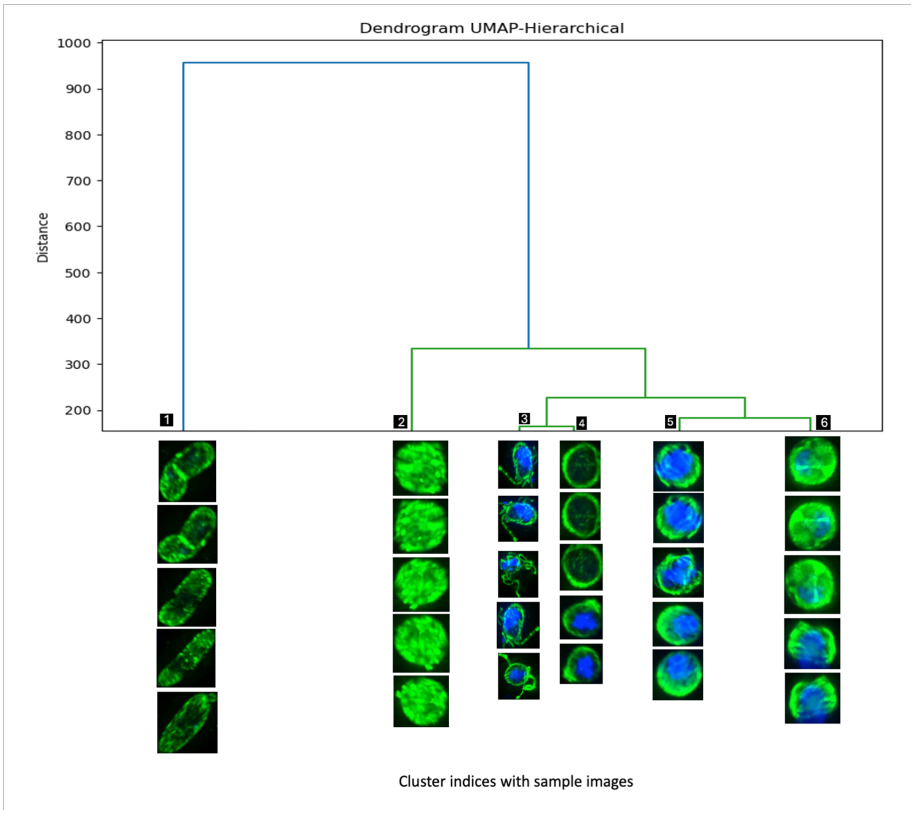


Figure 6.12: Dendrogram for UMAP-Hierarchical model with sample images from each cluster. It is observed from cluster 4 and 6 that they are mixed with arrested level mid 1 and mid 2 cells.

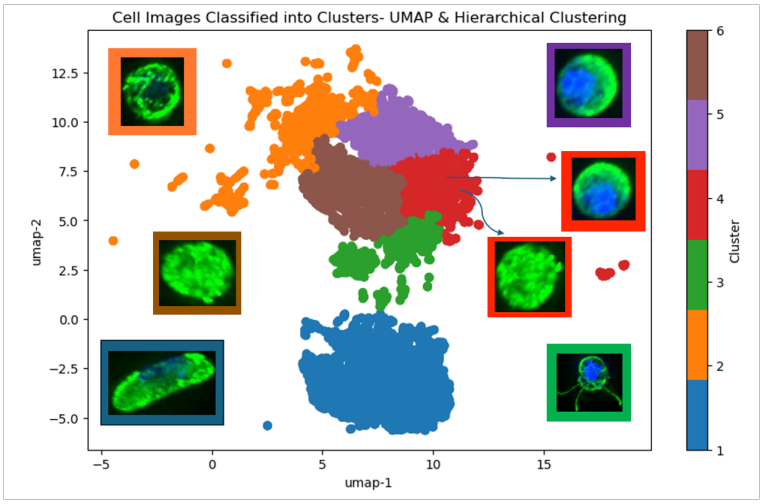


Figure 6.13: UMAP-Hierarchical clusters with a sample image from each cluster.

Evaluation Metrics	**Score**
Average Majority Score	58.03%
Silhouette Score	0.416580991
Calinski-Harabasz Index	25387.17107
Davies-Bouldin Index	0.833853115

Table 6.10: Clustering Evaluation Metrics for UMAP-Hierarchical model.

6.2.8 Summary

- It is evident from the experiments that lower dimensional clustering models produced less overlapping clusters.
- The arrested early coffee bean-shaped cells are able to be identified as a separate cluster by UMAP. In contrast, every cluster in an Isomap remains attached to every other cluster.
- The exflagellated cells can be clustered independently using the UMAP-Hierarchical model.
- While Kmeans models produced good metric scores, hierarchical models offered the clusters exact boundaries.
- Even though overlapping problems were avoided by low dimensional clustering models, cluster boundaries stay attached to one another.
- The UMAP-Hierarchical model's cluster 5's arrested mid level 2 cells differ slightly from cluster 6 in the model(refer **Figure 6.12**). Instead of having typical spherical cells, some of the cells in cluster 5 exhibited ellipse-shaped cells with light blue nuclei proving the presence of fuzzy patterns.

Chapter 7

Setting Benchmark Score to the Models

All of the experiments' metrics for evaluation are collected and compared with one another. Different models have the greatest scores in each metric. For instance, the tSNE-Kmeans model has the largest average majority share, while the Isomap-Kmeans model gets the highest silhouette score. This makes it difficult to select the most suitable model for the dataset. In order to make it easier to select the best model, WSM (Weighted Sum Method)(MacCrimmon 1968), one of the multi attribute techniques, has been used to set a single benchmark score for each model.

7.1 Initialising Weights for the Metrics

Initialising weight for each metric is the first stage in this process(MacCrimmon 1968). When choosing a model, weights serve as considerations for priority. Since choosing a well-separated model is the primary goal in this case, calinksi-harabaz is assigned a weight of 30% because it shows the ratio between inter cluster and intra cluster dispersion. Each of the Davies-Bouldin index and the silhouette score has been assigned a 25% weight as both of them include inter and intra cluster distances. Average majority score is given only 20% as textbook evaluation metrics are given more preference.

Metric	**Weight**	**tSNE-Kmeans(PhIDDLI)**	**UMAP - Kmeans**	**UMAP - Hierarchical**	**Isomap-Kmeans**	**Isomap-Hierarchical**
Average Majority Share	20%	65.9%	61.5%	58.03%	58.40%	57.90%
Silhouette Score	25%	0.08913	0.4642	0.416580991	0.473804784	0.412123176
Calinski-Harabasz Index	30%	4935.989	33310.15	25387.17107	21305.62859	16593.17
Davies-Bouldin Index	25%	1.809	0.834	0.833853115	0.832624518	0.9105

Table 7.1: Comparison of Clustering Models Based on Various Metrics.

7.2 Normalisation

Now the values are normalised to make the benchmark score between 0 to 1. The lowest value among the models is regarded as the best value for the Davies-Bouldin Index, therefore the values are inversed. Now the updated table is given as:

Metric	Weight	tSNE-Kmeans(PhIDDLI)	UMAP - Kmeans	UMAP - Hierarchical	Isomap-Kmeans	Isomap-Hierarchical
Average Majority Share	20%	65.9%	61.5%	58.03%	58.40%	57.90%
Silhouette Score	25%	0.08913	0.4642	0.416580991	0.473804784	0.412123176
Calinski-Harabasz Index	30%	4935.989	33310.15	25387.17107	21305.62859	16593.17
Davies-Bouldin Index(Inverted)	25%	0.553	1.199	1.200	1.201	1.099

Table 7.2: Updated Table with Davies Bouldin Index inverted.

Now the table is normalised using min max normalisation(Vafaei et al. 2018). The formula for min max normalisation is given below:

$$X' = \frac{X - \min(X)}{\max(X) - \min(X)} \tag{7.1}$$

Where,

- X' is the normalised value.
- min(X) is the minimum value in the dataset.
- man(X) is the maximum value in the dataset.

A sample calculation for normalising silhouette score for PhIDDLI model is given below:

$$\text{Normalised Silhouette Score}_{PhIDDLI} = \frac{0.08913 - 0.08913}{0.384674784} = \frac{0.0}{0.384674784} = 0.0$$

Now all the values are normalised and the normalised table is given below:

Metric	Weight	tSNE-Kmeans (PhIDDL)	UMAP - Kmeans	UMAP - Hierarchical	Isomap-Kmeans	Isomap-Hierarchical
Average Majority Share	20%	1.0	0.45	0.01625	0.0625	0.0
Silhouette Score	25%	0.0	0.975	0.852	1.0	0.839
Calinski-Harabasz Index	30%	0.0	1.0	0.72	0.577	0.411
Davies-Bouldin Index(Inverted)	25%	0.0	0.998	1.0	1.0	0.843

Table 7.3: Normalised Evaluation Metrics Table.

7.3 Applying the weights

The final step involves applying the weights on the normalised values to get a single benchmark score for each value. The final equation to calculate the weighted score(MacCrimmon 1968):

$$\text{Weighted Score}_i = \sum_{j=1}^{n} w_j \cdot x_{ij} \tag{7.2}$$

where:

$$\begin{aligned} \text{Weighted Score}_i &: \text{The benchmark score for each model } i, \\ w_j &: \text{The weight assigned to the metric } j, \\ x_{ij} &: \text{The normalized value of metric } j \text{ for model } i, \\ n &: \text{The total number of metrics.} \end{aligned}$$

A sample calculation of the weighted benchmark score for tSNE-Kmeans model is given below:

$$\begin{aligned} \text{Weighted Score for tSNE-Kmeans model} &= (0.0 \times 0.25) + (0.0 \times 0.30) \\ &\quad + (0.0 \times 0.30) + (1.000000 \times 0.20) \\ &= 0.20 \end{aligned}$$

Similarly the weighted benchmark score for rest of the models are calculated and plotted below:

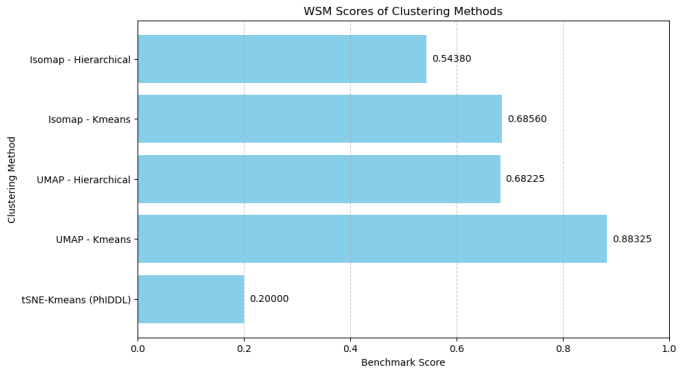


Figure 7.1: WSM results for clustering models.

From the plot it is understood that UMAP-Kmeans model gets the highest benchmark score compared to others and considered as the best model for the dataset. This proves that the exponential distribution(McInnes et al. 2018)—rather than the gaussian kernel(Van Der Maaten & Hinton 2008) or shortest path(The Greedy Choice 2017)—was a more successful method for calculating the similarity score for the data points in this dataset. Additionally, for this dataset, employing the entropy loss function(McInnes et al. 2018) has been more successful than KL divergence(Van Der Maaten & Hinton 2008) or multidimensional scaling(The Greedy Choice 2017). This means UMAP data points are better for modeling than Isomap and t-SNE data points.

Chapter 8

Improvising the Model

The images on the boundaries are difficult to diagnose since they are sticking together, even after the best model from the experiments is chosen. The UMAP-Kmeans model primarily consists of four clusters: arrested early, mid 1, mid 2, and arrested late. Nonetheless, clustering efforts have already assisted in labelling these groupings. There are situations in which an image may come from partially one class and partially another. The Kmeans model can be modified to the fuzzy c-means model(Pedrycz 1996) in order to find the fuzzy and hidden patterns. Also the unsupervised model can be modified to semi supervised model(Pedrycz 1996) with a subset of labelled data points.

8.1 Semi Supervised Clustering

A small subset of data points from the image embeddings has been labelled to act as anchor points that can guide the clustering for the rest of the unlabelled data points(Pedrycz 1996).

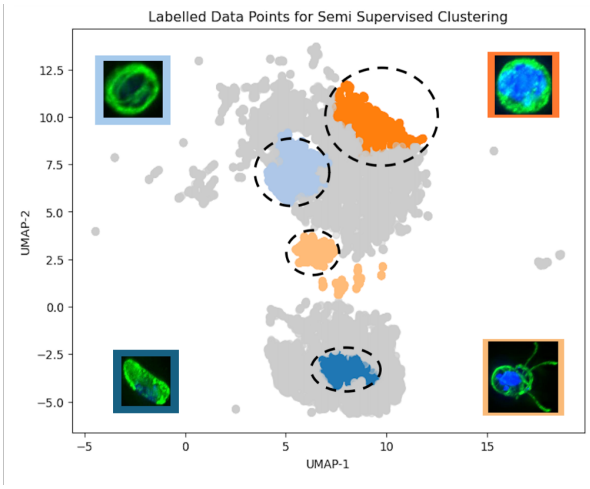


Figure 8.1: Labelled anchor points for semi supervised model with a sample image from each class.

8.2 Outliers

The data points that represented unclear/noisy images has been identified and removed form the UMAP structure. The corresponding data points are highlighted in the image below.

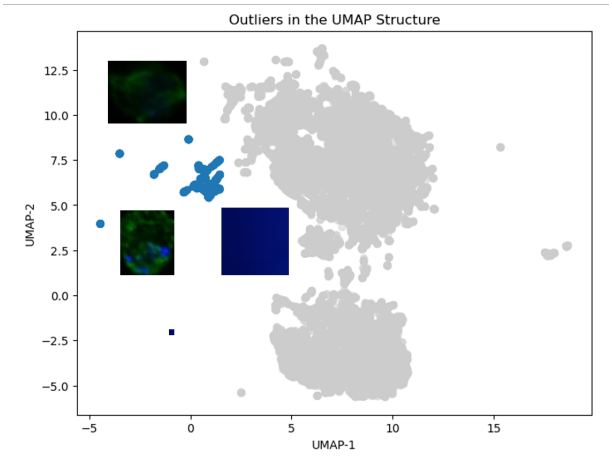


Figure 8.2: Noisy data points with sample images from them.

8.3 Fuzzy C-means Clustering

Fuzzy C-means clustering allows each data point to belong to more than one cluster by using membership vectors(Bezdek et al. 1984). The algorithm is an expansion of the K-means algorithm. In the K-means algorithm, each data point is assigned to a specific cluster. In the fuzzy c-means algorithm, each data point is associated with a list of probabilities indicating its membership in each cluster. The first step is to find the optimal number of clusters(k) which is almost same as K-means model. In this dataset, the optimal number of clusters is set as four as this is an upgrade of UMAP-Kmeans model. The next step involves finding the centroids of the clusters(Bezdek et al. 1984).

$$v_j = \frac{\sum_{i=1}^{n} u_{ij}^m \cdot x_i}{\sum_{i=1}^{n} u_{ij}^m} \tag{8.1}$$

Where:

v_j is the centroid of cluster j.

u_{ij} is the membership degree of data point x_i in cluster j.

m is the fuzziness parameter.

x_i is the i-th data point.

At first membership values and the the centroids are assigned randomly but in this case anchor points will influence the initialisation(Pedrycz 1996). The fuzziness parameter (m) is utilised to precisely determine the equal distribution of membership values for the data points. When m equals 1, the model will function as a typical hard clustering K-means model.If the value increases, all of the membership values for a given data point will be equal, indicating that the data point equally belongs to all the clusters. In this model, fuzziness parameter is set as 3. The next step involves updating the membership matrix. The equation for updating the membership matrix is given as(Bezdek et al. 1984):

$$\hat{u}_{ik} = \left(\sum_{j=1}^{c} \left(\frac{d_{jk}}{d_{ik}} \right)^{\frac{2}{m-1}} \right)^{-1} \tag{8.2}$$

Where:

$\hat{u}_{ik}$: Updated membership value of the k-th data point in the i-th cluster.

d_{ik} : Distance between the k-th data point and the i-th cluster center.

d_{jk} : Distance between the k-th data point and the j-th cluster center.

c : Number of clusters.

m : Fuzziness parameter.

The membership matrix and cluster centroids are initialised and updated repeatedly until convergence is achieved.

8.4 Semi Supervised Fuzzy C-means Clustering

The objective function for the semi supervised fuzzy C-means clustering is(Pedrycz 1996):

$$Q = \sum_{i=1}^{c} \sum_{k=1}^{N} u_{ik}^2 d_{ik}^2 + \alpha \sum_{i=1}^{c} \sum_{k=1}^{N} (u_{ik} - f_{ik} b_k)^2 d_{ik}^2 \tag{8.3}$$

- **First Term:** Represents the usual clustering objective, where u_{ik} is the membership degree of pattern k in cluster i and d_{ik} is the distance between pattern k and the centroid of cluster i.

- **Second Term:** Adjusts for partial supervision by incorporating f_{ik}, the membership grade based on labels, and b_k, a binary variable indicating whether pattern k is labeled.

- **Alpha (α):** A parameter controlling the influence of the labeled patterns on the clustering process. In this model, the value of (α) is set as 0.5.

The objective function makes sure that the appropriate clustering configuration has been chosen for optimisation. To optimise, the algorithm seeks to minimise the objective function.

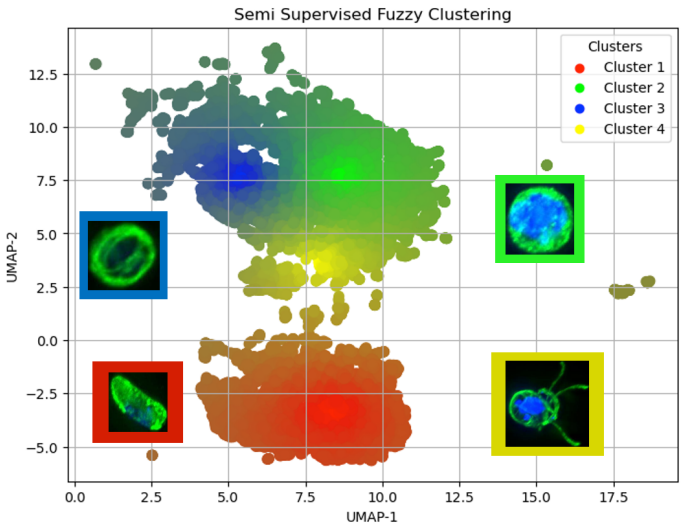


Figure 8.3: Visualisation of Semi Supervised Fuzzy Clusters with a sample image from each class.

8.5 Analysing the Fuzzy Patterns

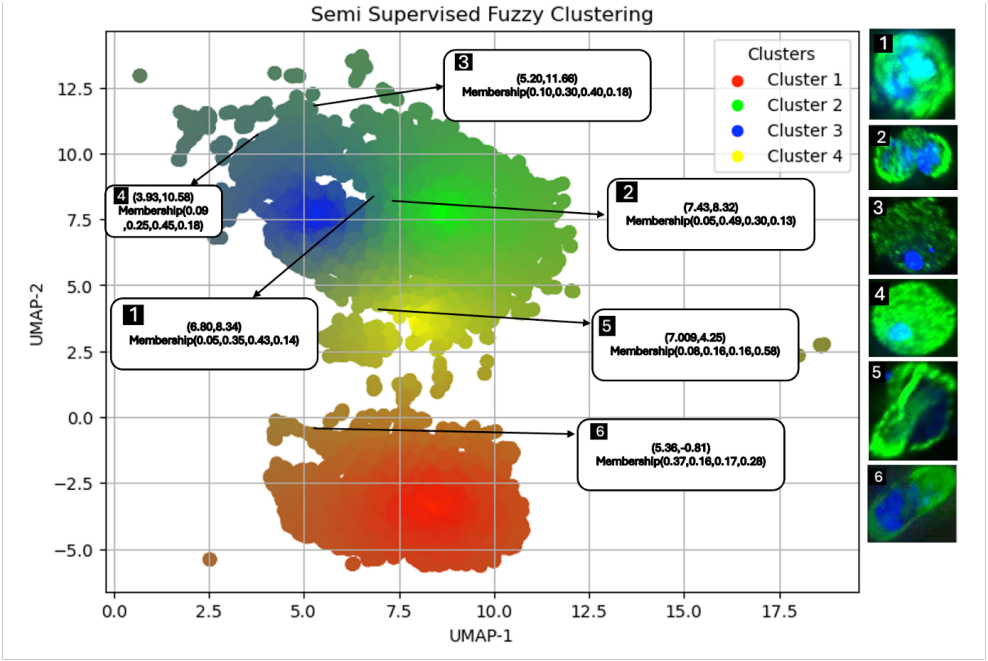


Figure 8.4: Data Points with coordinates with their membership vectors and numbered with their respective images.

8.5.1 Image 1: Gametocyte with a Pale Blue Nucleus

According to the algorithm, image 1 in **Figure 8.4**, it falls into the 35% arrested mid level 2 and 43% arrested level mid 1 category. The ideal nucleus would have a concentrated a blue color, but the image shows a pale blue nucleus dispersed throughout the rounded gametocyte cell. The TC57 drug has assisted in achieving the outcome shown in image 1.

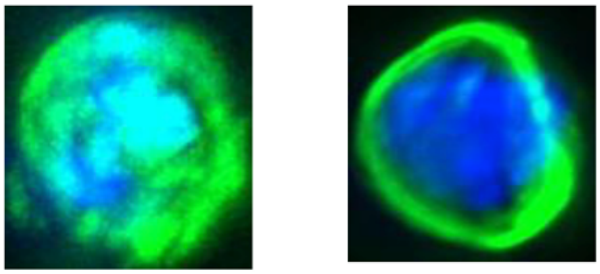


Figure 8.5: Comparison of nucleus in the image of interest vs ideal nucleus.

8.5.2 Image 2: Multiplet Cells

Image 2 illustrates how an arrested mid-level 2 cell divides into two cells, each of which has the potential to evolve into an exflagellation. Multiplet cell is the term for this stage. According to the algorithm, the cell belongs to 30% of arrested mid level 1, 15% of arrested late, and 49% of arrested mid level 2. The TC880 drug has helped us reach this point. Biologists and researchers can look at how the drug and cell interacted to get to this point in more detail.

8.5.3 Image 3: Gametocyte with Polarised Nucleus

There is a tiny, focused blue nucleus in Image 3. It is absent from the middle, though. Rather, the nucleus is polarized in proximity to the cell's membrane wall. According to the algorithm, 40% of arrested individuals belong to arrested mid level 1, and 30% belong to arrested mid level 2. This stage has been attained with the use of MB (Methylene Blue).

8.5.4 Image 4: Gametocyte with Polarised Pale Blue Nucleus

The properties of image 3 are the same as those of image 4, but the nucleus is light blue rather than dark blue. 25% of the cells are projected by the algorithm to belong to arrested mid level 2, and 45% to arrested mid level 1. At this point, a medication known as TC02 has stopped the cell.

8.5.5 Image 5: Semi Exflagellated cell

A cell in Image 5 has stopped halfway between mid-level and exflagellation. It is evident that the cell had begun to evolve but was inadvertently stopped in its tracks by the drug. The cell is expected to belong to 16% mid level, 16% mid level 2 and 58% arrested late, according to the algorithm. This outcome is attributable to the MMV73 drug, a particular molecule from MMV (Medicines for Malaria Venture).

8.5.6 Image 6: Coffee Bean with a nucleus

A coffee bean-shaped early stage gametocyte with a developing nucleus is seen in Image 6. This is a special stage since normally the gametocyte develops into a rounded cell and the nucleus starts to form. Here, however, the nucleus develops before the cell becomes rounded. According to the algorithm, the cell will have 37% belonging to arrested early, 16% to mid level 1, 17% to mid level 2, and 28% to arrested late. It was a solvent named DMSO (dimethyl sulfoxide) that helped achieve this outcome.

Chapter 9

Epilogue

9.1 Further Area of Study

There is still potential for improvement in cell detection even though dimensionality reduction and clustering strategies have been investigated through several experiments and the model has been optimized to uncover hidden fuzzy patterns. At the moment, the ICY platform is used to manually detect the cells. Techniques like object detection(Zhao et al. 2019) and image segmentation(Minaee et al. 2022) can be used to automate this procedure. Moreover, the anticipated images using the membership vectors and the model can be expanded to directly interpret the medications. For now, the analysis of the drugs is done manually. These actions can significantly improve the pipeline's ability to function effectively.
Furthermore, the dataset's features can be learned using an autoencoder network(Chen et al. 2021). Using a Mean Squared Error loss function as a gauge, autoencoders attempt to minimize the difference between the original dataset and the reconstructed data points. If the auto encoder is successful in reconstructing the data points with as little loss as possible, then this method can fix the sticking borders problem.

9.2 Final Discussion

The lifecycle of malaria has been discovered and the background survey on the primary screen dataset has been completed. We now know the stages of plasmodium gametocyte cells' evolution. Analysis has been done on the PhIDDLI pipeline's steps. After the pipeline model is diagnosed, it is discovered that the model has issues with sticking and overlapping borders. Additionally, the model is lagging in evaluation metrics including the Calinski Harabaz index, Davies Bouldin index, and silhouette score. To assess how well the images fit within each cluster, a customized CNN model was created. To demonstrate the curse of dimensionality, the dataset is put to the test. Then, in order to reduce the dimensionality of Isomap and UMAP, clustering experiments were

carried out. A tiny portion of the data has been used to observe and illustrate how Isomap and UMAP operate. Kmeans and Hierarchical clustering have been used to apply and model each dimensionality reduction. It has been noted that UMAP does a superior job of separating the clusters. Furthermore, the overlapping problem with cluster boundaries has been resolved with the use of low dimensional clustering.Even so, some boundaries continue to be sticking together. The benchmark score for each model was established using a multi-attribute technique known as the weighted sum method, and it was discovered that UMAP models outperform Isomap and PhIDDLI models. The anchor points for a semi-supervised model have been found with the use of UMAP models, and fuzzy clustering has been applied to the UMAP structure. In order to demonstrate the value of the semi-supervised fuzzy clustering technique, a few fuzzy hidden patterns have been found. Researchers studying medicine can use the final model to identify deep, hidden patterns in the gametocyte cells and uncover ways to create more potent drugs that stop malaria transmission.

Bibliography

Abdi, H. & Williams, L. J. (2010), 'Principal component analysis', *Wiley Interdisciplinary Reviews: Computational Statistics* **2**(4), 433–459.

Altman, N. & Krzywinski, M. (2018), 'The curse(s) of dimensionality', *Nature Methods* **15**(6), 399–400.

Bezdek, J. C., Ehrlich, R. & Full, W. (1984), 'Fcm: The fuzzy c-means clustering algorithm', *Computers & Geosciences* **10**(2-3), 191–203.

Calinski, T. & Harabasz, J. (1974), 'A dendrite method for cluster analysis', *Communications in Statistics* **3**(1), 1–27.

Carroll, J. D. & Arabie, P. (1998), Multidimensional scaling, *in* M. H. Birnbaum, ed., 'Handbook of Perception and Cognition (Second Edition), Measurement, Judgment and Decision Making', Academic Press, San Diego, pp. 179–250.
URL: *https://www.sciencedirect.com/science/article/pii/B9780120999750500051*

Chen, M., Shi, X., Zhang, Y., Wu, D. & Guizani, M. (2021), 'Deep feature learning for medical image analysis with convolutional autoencoder neural network', *IEEE Transactions on Big Data* **7**(4), 750–758.

Davie s, D. L. & Bouldin, D. W. (1979), 'A cluster separation measure', *IEEE Transactions on Pattern Analysis and Machine Intelligence* **PAMI-1**(2), 224–227.

Ghodsi, A. (2006), 'Dimensionality reduction a short tutorial', *Department of Statistics and Actuarial Science, Univ. of Waterloo, Ontario, Canada* **37**(38), 2006.

Goodfellow, I., Bengio, Y. & Courville, A. (2016), *Deep Learning*, MIT press.

Kingma, D. P. & Ba, J. (2014), 'Adam: A method for stochastic optimization', *arXiv preprint arXiv:1412.6980* pp. 1–15.

LeCun, Y., Bottou, L., Bengio, Y. & Haffner, P. (1998), 'Gradient-based learning applied to document recognition', *Proceedings of the IEEE* **86**(11), 2278–2324.

MacCrimmon, K. R. (1968), Decision making among multiple-attribute alternatives: A survey and consolidated approach, *in* 'Multiple Criteria Decision Making', Rand Corporation, Santa Monica, CA, pp. 25–47.

MacQueen, J. (1967), Some methods for classification and analysis of multivariate observations, *in* 'Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics', University of California Press, pp. 281–297.
URL: *https://projecteuclid.org/euclid.bsmsp/1200512992*

McInnes, L., Healy, J. & Melville, J. (2018), 'UMAP: Uniform manifold approximation and projection for dimension reduction', *arXiv preprint arXiv:1802.03426* .
URL: *https://arxiv.org/abs/1802.03426*

Minaee, S., Boykov, Y., Porikli, F., Plaza, A., Kehtarnavaz, N. & Terzopoulos, D. (2022), 'Image segmentation using deep learning: A survey', *IEEE Transactions on Pattern Analysis and Machine Intelligence* **44**(7), 3523–3542.

Nocedal, J. & Wright, S. J. (2006), *Numerical Optimization*, 2nd edn, Springer, New York.

Organization, W. H. (2000), *Management of severe malaria: a practical handbook*, World Health Organization.

Pedrycz, W. (1996), 'Fuzzy clustering with a knowledge-based guidance', *Pattern Recognition Letters* **17**(6), 87–96.

Quizlet (2024), 'Advanced bio: Plasmodium life cycle diagram', `https://quizlet.com/ca/274351701/advanced-bio-plasmodium-life-cycle-diagram/`. Accessed: 2024-08-18.

Risald, A. E. M. & Suyoto (2017), Best routes selection using dijkstra and floyd-warshall algorithm, *in* '2017 11th International Conference on Information & Communication Technology and System (ICTS)', Surabaya, Indonesia, p. 156.

Rousseeuw, P. J. (1987), 'Silhouettes: A graphical aid to the interpretation and validation of cluster analysis', *Journal of Computational and Applied Mathematics* **20**, 53–65.

The Greedy Choice (2017), 'Isomap algorithm steps and example'. Accessed: 2024-07-28.
URL: *https://thegreedychoice.github.io/isomap-algorithm-steps*

Tomašev, N. & Radovanović, M. (2016), Clustering evaluation in high-dimensional data, *in* M. Celebi & K. Aydin, eds, 'Unsupervised Learning Algorithms', Springer, Cham, pp. 87–121.

Tsebriy, O., Khomiak, A., Miguel-Blanco, C., Sparkes, P. C., Gioli, M., Santelli, M., Whitley, E., Gamo, F.-J. & Delves, M. J. (2023), 'Machine learning-based phenotypic imaging to characterise the targetable biology of plasmodium falciparum male gametocytes for the development of transmission-blocking antimalarials', *bioRxiv* .
URL: *https://doi.org/10.1101/2023.04.22.537818*

Vafaei, N., Ribeiro, R. A. & Camarinha-Matos, L. M. (2018), 'Data normalisation techniques in decision making: case study with topsis method', *International Journal of Information and Decision Sciences* **10**(1), 1137.

Van Der Maaten, L. & Hinton, G. (2008), 'Visualizing data using t-sne', *Journal of Machine Learning Research* **9**, 2579–2605.
URL: *http://www.jmlr.org/papers/volume9/vandermaaten08a/vandermaaten08a.pdf*

Wikipedia (2024), 'Floyd–warshall algorithm — wikipedia, the free encyclopedia', `https://en.wikipedia.org/wiki/Floyd%E2%80%93Warshall_algorithm#Pseudocode`. Accessed: 2024-08-21.

World Health Organization (2023), 'Malaria: Life cycle of the malaria parasite'. Accessed: 22 July 2024.
URL: *https://www.who.int/news-room/fact-sheets/detail/malaria*

Xu, R. & Wunsch, D. C. (2010), 'Clustering algorithms in biomedical research: a review', *IEEE Reviews in Biomedical Engineering* **3**, 120–154.

Zhao, Z.-Q., Zheng, P., Xu, S.-T. & Wu, X. (2019), 'Object detection with deep learning: A review', *IEEE Transactions on Neural Networks and Learning Systems* **30**(11), 3212–3232.

Appendix

September 2, 2024

For accessing the PhIDDLI model visit the following link:https://github.com/michaeldelves/PhIDDLI/tree/main

Challenges in High Dimensional Clustering

Observation of Evaluation Metrics over the increase in dimensions

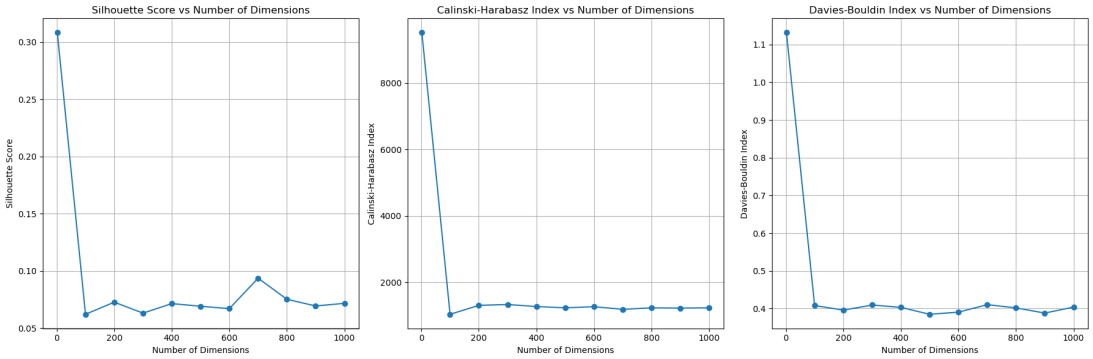
[5]:
```
import numpy as np
import pandas as pd
from sklearn.neighbors import NearestNeighbors
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score, calinski_harabasz_score,␣
 ↪davies_bouldin_score
import matplotlib.pyplot as plt
import random

# Load the high-dimensional dataset from a CSV file
def load_data_from_csv(file_path):
    data = pd.read_csv(file_path)
    data=data.T
    return data

# Select a subset of dimensions randomly
def select_random_dimensions(data, n_dimensions):
    columns = random.sample(list(data.columns), n_dimensions)
    return data[columns].values

# Compute neighbor occurrence frequency (Hubness)
def compute_hubness(X, k=10):
    nbrs = NearestNeighbors(n_neighbors=k).fit(X)
    distances, indices = nbrs.kneighbors(X)
    neighbor_counts = np.zeros(X.shape[0])
    for idx in indices:
        neighbor_counts[idx] += 1
    return neighbor_counts

# Cluster the dataset and return labels
def cluster_data(X, n_clusters=15):
    kmeans = KMeans(n_clusters=n_clusters, random_state=42)
    labels = kmeans.fit_predict(X)
```

```
    return labels

# Main function to run the analysis
def main():
    # Load data from CSV file
    file_path = '/Users/vishanthsuresh/Downloads/Data Science/Dissertation/
 ↪PhIDDLI-main/data/embeddings_aug_B1_primary.csv'  # Replace with your actual␣
 ↪file path
    data = load_data_from_csv(file_path)

    # List of dimensions to iterate over
    dimensions_list = [2] + list(range(100, 1100, 100))

    silhouette_scores = []
    calinski_harabasz_scores = []
    davies_bouldin_scores = []

    for n_dimensions in dimensions_list:
        # Select random dimensions
        X = select_random_dimensions(data, n_dimensions)

        # Compute hubness
        neighbor_counts = compute_hubness(X)

        # Cluster data
        labels = cluster_data(X)

        # Calculate silhouette score
        silhouette = silhouette_score(X, labels)
        silhouette_scores.append(silhouette)

        # Calculate Calinski-Harabasz Index
        calinski_harabasz = calinski_harabasz_score(X, labels)
        calinski_harabasz_scores.append(calinski_harabasz)

        # Calculate Davies-Bouldin Index
        davies_bouldin = 1/davies_bouldin_score(X, labels)
        davies_bouldin_scores.append(davies_bouldin)

        print(f"Dimensions: {n_dimensions}, Silhouette Score: {silhouette},␣
 ↪Calinski-Harabasz Index: {calinski_harabasz}, Davies-Bouldin Index:␣
 ↪{davies_bouldin}")

    # Plot the scores
    plt.figure(figsize=(18, 6))

    # Silhouette Score
```

```
    plt.subplot(1, 3, 1)
    plt.plot(dimensions_list, silhouette_scores, marker='o')
    plt.xlabel('Number of Dimensions')
    plt.ylabel('Silhouette Score')
    plt.title('Silhouette Score vs Number of Dimensions')
    plt.grid(True)

    # Calinski-Harabasz Index
    plt.subplot(1, 3, 2)
    plt.plot(dimensions_list, calinski_harabasz_scores, marker='o')
    plt.xlabel('Number of Dimensions')
    plt.ylabel('Calinski-Harabasz Index')
    plt.title('Calinski-Harabasz Index vs Number of Dimensions')
    plt.grid(True)

    # Davies-Bouldin Index
    plt.subplot(1, 3, 3)
    plt.plot(dimensions_list, davies_bouldin_scores, marker='o')
    plt.xlabel('Number of Dimensions')
    plt.ylabel('Davies-Bouldin Index')
    plt.title('Davies-Bouldin Index vs Number of Dimensions')
    plt.grid(True)

    plt.tight_layout()
    plt.show()

if __name__ == "__main__":
    main()
```

```
Dimensions: 2, Silhouette Score: 0.30821960097749607, Calinski-Harabasz Index:
9521.070155469304, Davies-Bouldin Index: 1.131343338640496
Dimensions: 100, Silhouette Score: 0.061971248215102366, Calinski-Harabasz
Index: 1032.5532715834659, Davies-Bouldin Index: 0.4082109198516956
Dimensions: 200, Silhouette Score: 0.07260070029612507, Calinski-Harabasz Index:
1302.4719463052986, Davies-Bouldin Index: 0.3959735044997108
Dimensions: 300, Silhouette Score: 0.06300499460989602, Calinski-Harabasz Index:
1328.2768845345158, Davies-Bouldin Index: 0.40958132983860834
Dimensions: 400, Silhouette Score: 0.07133252458699957, Calinski-Harabasz Index:
1266.6843030135153, Davies-Bouldin Index: 0.4032594446699761
Dimensions: 500, Silhouette Score: 0.06901342206791014, Calinski-Harabasz Index:
1228.6291928458697, Davies-Bouldin Index: 0.38489365075105264
Dimensions: 600, Silhouette Score: 0.06692588827454864, Calinski-Harabasz Index:
1261.333191801465, Davies-Bouldin Index: 0.39067155735805337
Dimensions: 700, Silhouette Score: 0.09357897841715351, Calinski-Harabasz Index:
1183.7494255766837, Davies-Bouldin Index: 0.4104643810879705
Dimensions: 800, Silhouette Score: 0.07511850105351464, Calinski-Harabasz Index:
1226.3294095765943, Davies-Bouldin Index: 0.4019582877465828
Dimensions: 900, Silhouette Score: 0.06927039780949519, Calinski-Harabasz Index:
```

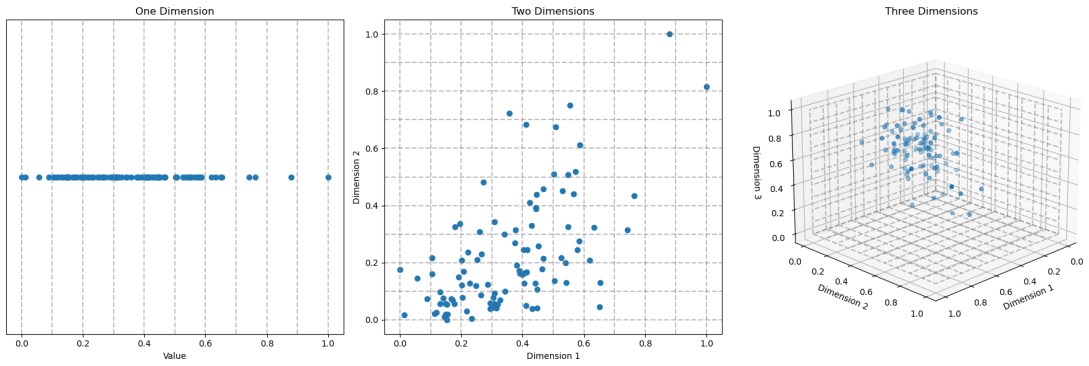
1220.3707033531489, Davies-Bouldin Index: 0.3881637214843754
Dimensions: 1000, Silhouette Score: 0.07156348938036947, Calinski-Harabasz Index: 1227.3582978178738, Davies-Bouldin Index: 0.40395099656107053



[2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import random

# Load the dataset
df = pd.read_csv('/Users/vishanthsuresh/Downloads/Data Science/Dissertation/
 ↪PhIDDLI-main/data/embeddings_aug_B1_primary.csv')
df = df.T

# Select 100 random points
random_indices = random.sample(range(len(df)), 100)
random_points = df.iloc[random_indices]

# Normalize the data to range [0, 1]
random_points = (random_points - random_points.min()) / (random_points.max() -␣
 ↪random_points.min())

# Function to discretize the space and count empty cells
def count_empty_cells(data, dimensions, cell_size):
    grid_shape = tuple(int(1 / cell_size) for _ in range(dimensions))
    grid = np.zeros(grid_shape)

    for point in data:
        indices = tuple(int(coord / cell_size) for coord in point[:dimensions])
        if all(0 <= index < size for index, size in zip(indices, grid_shape)):
            grid[indices] += 1

    empty_cells = np.sum(grid == 0)
    total_cells = grid.size
```

```
    return empty_cells, total_cells

# Creating a figure with three subplots in a single row with equal sizes
fig = plt.figure(figsize=(18, 6))  # Increased the height to make the plots␣
 ↪equal in size

# Plotting one dimension as a straight line with grid cells
ax1 = fig.add_subplot(131)
ax1.plot(random_points.iloc[:, 0], np.zeros(100), 'o')
ax1.set_title('One Dimension')
ax1.set_xlabel('Value')
ax1.set_yticks([])  # Hide y-axis
# Add grid lines
for x in np.arange(0, 1.1, 0.1):
    ax1.axvline(x, color='gray', linestyle='--', alpha=0.5)

# Plotting two dimensions with grid cells
ax2 = fig.add_subplot(132)
ax2.scatter(random_points.iloc[:, 0], random_points.iloc[:, 1])
ax2.set_title('Two Dimensions')
ax2.set_xlabel('Dimension 1')
ax2.set_ylabel('Dimension 2')
# Add grid lines
for x in np.arange(0, 1.1, 0.1):
    ax2.axvline(x, color='gray', linestyle='--', alpha=0.5)
    ax2.axhline(x, color='gray', linestyle='--', alpha=0.5)

# Plotting three dimensions with grid cells
ax3 = fig.add_subplot(133, projection='3d')
ax3.scatter(random_points.iloc[:, 0], random_points.iloc[:, 1], random_points.
 ↪iloc[:, 2])
ax3.set_title('Three Dimensions')
ax3.set_xlabel('Dimension 1')
ax3.set_ylabel('Dimension 2')
ax3.set_zlabel('Dimension 3')
ax3.view_init(elev=20., azim=45)  # Adjust the angle for better visualization
# Add grid lines
for x in np.arange(0, 1.1, 0.1):
    ax3.plot([x, x], [0, 0], [0, 1], color='gray', linestyle='--', alpha=0.5)
    ax3.plot([x, x], [0, 1], [0, 0], color='gray', linestyle='--', alpha=0.5)
    ax3.plot([0, 0], [x, x], [0, 1], color='gray', linestyle='--', alpha=0.5)
    ax3.plot([0, 1], [x, x], [0, 0], color='gray', linestyle='--', alpha=0.5)
    ax3.plot([0, 0], [0, 1], [x, x], color='gray', linestyle='--', alpha=0.5)
    ax3.plot([0, 1], [0, 0], [x, x], color='gray', linestyle='--', alpha=0.5)

plt.tight_layout()
plt.show()
```

```
# Discretizing the space and counting empty cells
cell_size = 0.1

# One dimension
empty_cells_1d, total_cells_1d = count_empty_cells(random_points.values, 1,␣
 ↪cell_size)
print(f"One Dimension: {empty_cells_1d} empty cells out of {total_cells_1d}")

# Two dimensions
empty_cells_2d, total_cells_2d = count_empty_cells(random_points.values, 2,␣
 ↪cell_size)
print(f"Two Dimensions: {empty_cells_2d} empty cells out of {total_cells_2d}")

# Three dimensions
empty_cells_3d, total_cells_3d = count_empty_cells(random_points.values, 3,␣
 ↪cell_size)
print(f"Three Dimensions: {empty_cells_3d} empty cells out of {total_cells_3d}")
```



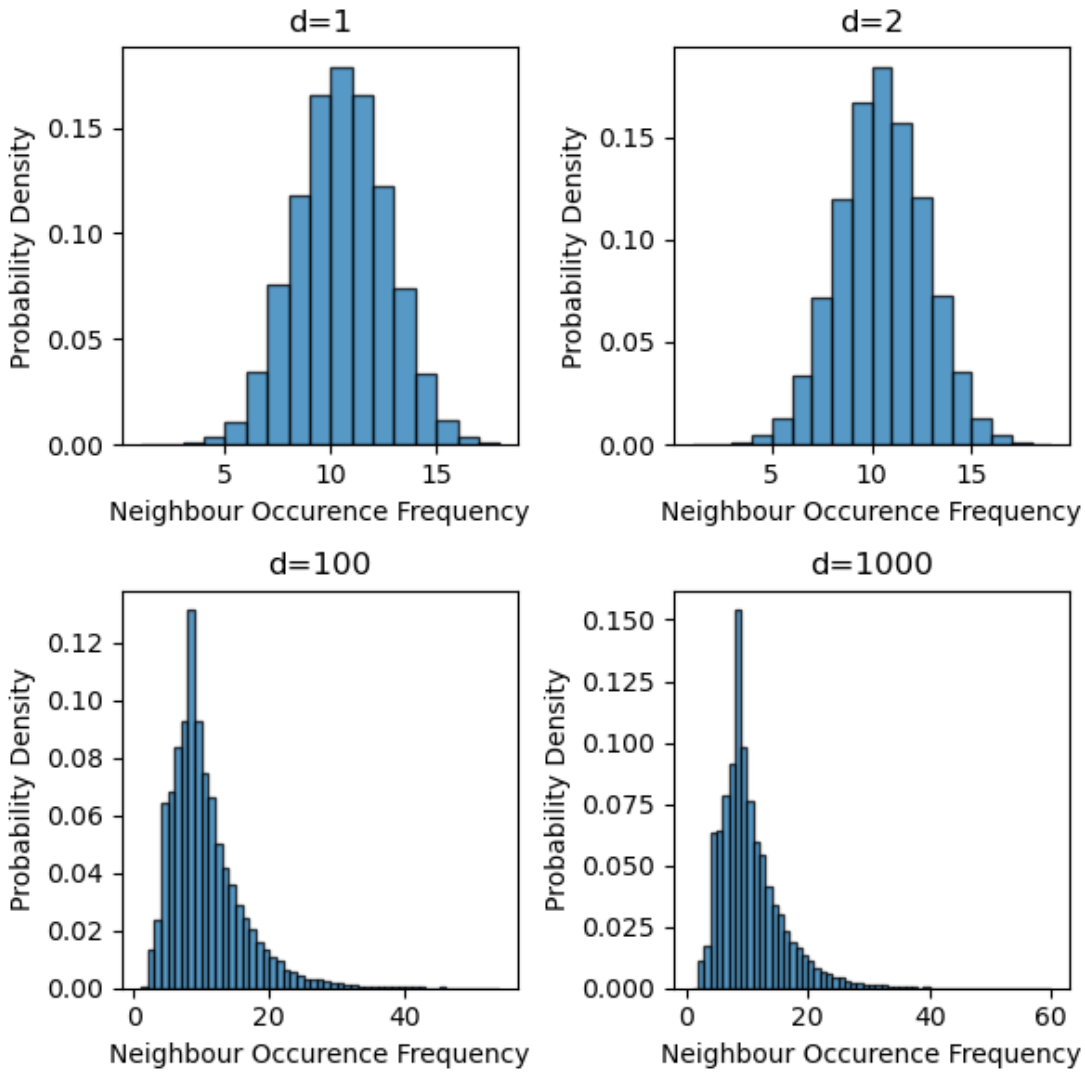

```
One Dimension: 1 empty cells out of 10
Two Dimensions: 65 empty cells out of 100
Three Dimensions: 928 empty cells out of 1000
```

1 Hubness effect

[57]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import NearestNeighbors

# Load the CSV dataset
dataset = pd.read_csv('/Users/vishanthsuresh/Downloads/Data Science/Dissertation/
 ↪PhIDDLI-main/data/embeddings_aug_B1_primary.csv')
```

```
dataset=dataset.T
# Function to calculate neighbor occurrence frequency
def neighbor_occurrence_frequency(data, k=10):
    nbrs = NearestNeighbors(n_neighbors=k).fit(data)
    _, indices = nbrs.kneighbors(data)
    occurrence_counts = np.zeros(len(data))

    for neighbors in indices:
        for neighbor in neighbors:
            occurrence_counts[neighbor] += 1

    return occurrence_counts

# Function to plot neighbor occurrence frequency distribution
def plot_distribution(ax, data, dimensions, k=10):
    occurrence_counts = neighbor_occurrence_frequency(data.iloc[:, :dimensions],␣
 ↪k)
    ax.hist(occurrence_counts, bins=range(1, max(occurrence_counts.astype(int))␣
 ↪+ 2), density=True, alpha=0.75, edgecolor='black')
    ax.set_title(f'd={dimensions}')
    ax.set_xlabel('Neighbour Occurence Frequency')
    ax.set_ylabel('Probability Density')

# Create a 2x2 grid of subplots
fig, axes = plt.subplots(2, 2, figsize=(6, 6))

# Single dimension
plot_distribution(axes[0, 0], dataset, dimensions=1)

# Two dimensions
plot_distribution(axes[0, 1], dataset, dimensions=2)

# 100 dimensions (assuming the dataset has at least 100 dimensions)
if dataset.shape[1] >= 100:
    plot_distribution(axes[1, 0], dataset, dimensions=100)
else:
    axes[1, 0].text(0.5, 0.5, 'Not enough dimensions',␣
 ↪horizontalalignment='center', verticalalignment='center', transform=axes[1, 0].
 ↪transAxes)

# 1000 dimensions (assuming the dataset has at least 1000 dimensions)
if dataset.shape[1] >= 1000:
    plot_distribution(axes[1, 1], dataset, dimensions=1000)
else:
    axes[1, 1].text(0.5, 0.5, 'Not enough dimensions',␣
 ↪horizontalalignment='center', verticalalignment='center', transform=axes[1, 1].
 ↪transAxes)
```

```
# Adjust layout and show the plots
plt.tight_layout()
plt.show()
```



[64]:
```
import numpy as np
import pandas as pd
from sklearn.neighbors import NearestNeighbors
from scipy.stats import skew

# Load the CSV dataset
```

```
dataset = pd.read_csv('/Users/vishanthsuresh/Downloads/Data Science/Dissertation/
 ↪PhIDDLI-main/data/embeddings_aug_B1_primary.csv')
dataset=dataset.T
# Function to calculate neighbor occurrence frequency
def neighbor_occurrence_frequency(data, k=10):
    nbrs = NearestNeighbors(n_neighbors=k).fit(data)
    _, indices = nbrs.kneighbors(data)
    occurrence_counts = np.zeros(len(data))

    for neighbors in indices:
        for neighbor in neighbors:
            occurrence_counts[neighbor] += 1

    return occurrence_counts

# Function to calculate skewness of the neighbor occurrence frequency␣
 ↪distribution
def calculate_skewness(data, dimensions, k=10):
    data_subset = data.iloc[:, :dimensions]
    occurrence_counts = neighbor_occurrence_frequency(data_subset, k)
    skewness_value = skew(occurrence_counts)
    return skewness_value

# Function to print skewness for given dimensions
def print_skewness_for_dimensions(dataset, dimensions, k=10):
    for dim in dimensions:
        if dataset.shape[1] >= dim:
            skewness_value = calculate_skewness(dataset, dim, k)
            print(f'Skewness for {dim} dimensions: {skewness_value}')
        else:
            print(f'The dataset does not have {dim} dimensions.')

# Specify the dimensions to be used
dimensions_to_test = [1, 2, 100, 1000]

# Print skewness for specified dimensions
print_skewness_for_dimensions(dataset, dimensions_to_test)
```

```
Skewness for 1 dimensions: -0.009939694892076769
Skewness for 2 dimensions: -0.019795075849741002
Skewness for 100 dimensions: 1.6608874957623627
Skewness for 1000 dimensions: 1.6948153769840806
```

2 Overall Result

From these experiments, reducing the dimensions can be proved as a way for better clustering

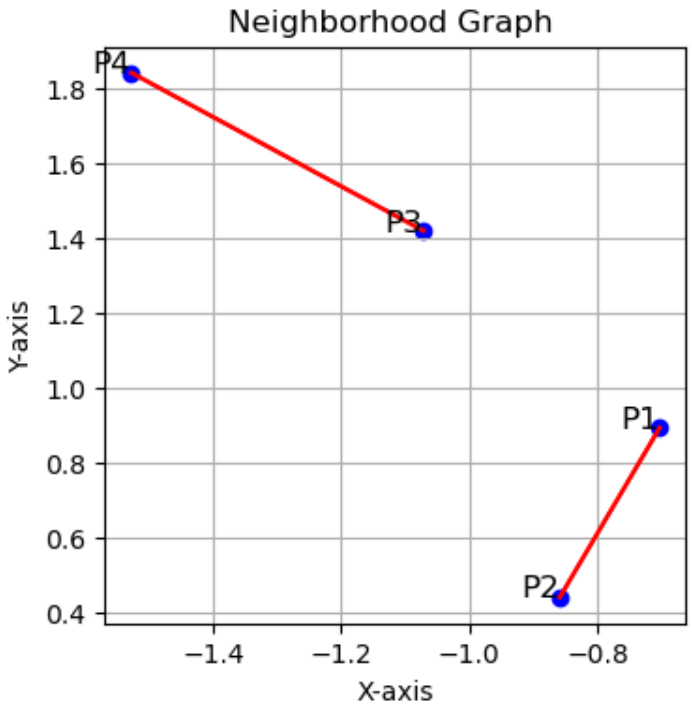
Isomap

September 2, 2024

0.0.1 Isomap Sample Calculation and Application

[24]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial import distance_matrix

# Given points
points = np.array([
    [-0.70323211,  0.89485216],
    [-0.85881805,  0.4431116 ],
    [-1.07164979,  1.42083216],
    [-1.52713239,  1.84265852]
])

# Calculate the distance matrix
distances = distance_matrix(points, points)

# Create a neighborhood graph where each point is connected to its nearest␣
 ↪neighbor
plt.figure(figsize=(4, 4))

# Plot the points
plt.scatter(points[:, 0], points[:, 1], color='blue')
for i, (x, y) in enumerate(points):
    plt.text(x, y, f'P{i+1}', fontsize=12, ha='right')

# Find and plot connections to nearest neighbors
for i in range(len(points)):
    # Ignore the distance to itself by setting it to infinity
    distances[i, i] = np.inf
    # Find the index of the nearest neighbor
    nearest_neighbor_idx = np.argmin(distances[i])
    # Plot the edge to the nearest neighbor
    plt.plot([points[i, 0], points[nearest_neighbor_idx, 0]],
             [points[i, 1], points[nearest_neighbor_idx, 1]], 'r-')

# Set plot title and labels
plt.title('Neighborhood Graph ')
```

```
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.grid(True)
plt.show()
```



[31]:
```
import numpy as np

# Define the matrix B
B = np.array([
    [-0.125388,  0.125388, -24.76043,  24.76043],
    [ 0.125388, -0.125388,  24.76043, -24.76043],
    [-24.76043,  24.76043, -0.125388,  0.125388],
    [ 24.76043, -24.76043,  0.125388, -0.125388]
])

# Perform eigen decomposition
eigenvalues, eigenvectors = np.linalg.eig(B)

# Sort the eigenvalues and eigenvectors by the absolute value of eigenvalues in␣
 ↪descending order
sorted_indices = np.argsort(-np.abs(eigenvalues))
eigenvalues = eigenvalues[sorted_indices]
eigenvectors = eigenvectors[:, sorted_indices]
```

```
# Select the principal eigenvector (corresponding to the largest eigenvalue)
principal_eigenvector = eigenvectors[:, 0]

# Normalize the principal eigenvector (if needed)
principal_eigenvector = principal_eigenvector / np.linalg.
 ↪norm(principal_eigenvector)

# The 1D embeddings are the entries of the normalized principal eigenvector
embedding_1D = principal_eigenvector

# Print the 1D embeddings
print("1D Embeddings:", embedding_1D)
```

```
1D Embeddings: [-0.5  0.5 -0.5  0.5]
```

[ ]:
```
import pickle
from argparse import ArgumentParser
from pathlib import Path
from typing import List, Tuple
import time

import numpy as np
from loguru import logger
from tqdm import tqdm
from sklearn.manifold import Isomap

# Function to apply ISOMAP transformation
def apply_isomap(vectors: np.ndarray, n_neighbors: int = 1, n_components: int =␣
 ↪2) -> np.ndarray:
    logger.info(f"Applying ISOMAP with {n_neighbors} neighbors and␣
 ↪{n_components} components")
    start_time = time.time()
    isomap = Isomap(n_neighbors=n_neighbors, n_components=n_components)
    result = isomap.fit_transform(vectors)
    end_time = time.time()
    time_taken = end_time - start_time
    logger.info(f"ISOMAP took {time_taken:.2f} seconds")
    return result

def load_embeddings(file: Path) -> Tuple[List[Path], np.ndarray]:
    embeddings = pickle.loads(file.read_bytes())
    files, vectors = zip(*list(embeddings.items()))
    vectors = np.array(vectors)  # Convert the list of vectors to a numpy array
    return files, vectors
```

```
def reduce_dimensionality(embeddings_file: Path, output_file: Path, n_neighbors:␣
↪int = 15, n_components: int = 2):
    files, vectors = load_embeddings(file=embeddings_file)
    points = apply_isomap(vectors=vectors, n_neighbors=n_neighbors,␣
↪n_components=n_components)
    points_mapping = dict(zip(files, points))
    output_file.write_bytes(pickle.dumps(points_mapping))

def set_paths_and_run(embeddings_file: str, output_file: str, n_neighbors: int =␣
↪15, n_components: int = 2):
    reduce_dimensionality(
        embeddings_file=Path(embeddings_file),
        output_file=Path(output_file),
        n_neighbors=n_neighbors,
        n_components=n_components
    )

if __name__ == "__main__":
    # Example programmatic usage
    embeddings_file = "/Users/vishanthsuresh/Downloads/Data Science/Dissertation/
↪PhIDDLI-main/data/embeddings_aug_B1_primary.pkl"
    output_file = "/Users/vishanthsuresh/Downloads/Data Science/Dissertation/
↪PhIDDLI-main/data/reduced_isomap_1.pkl"

    set_paths_and_run(embeddings_file, output_file)

    # Uncomment the following lines if you want to use command-line arguments␣
↪instead
    # parser = ArgumentParser()
    # parser.add_argument('embeddings_file', type=Path)
    # parser.add_argument('--output-file', type=Path, required=True)
    # parser.add_argument('--n_neighbors', type=int, default=5, help='Number of␣
↪neighbors to consider for each point')
    # parser.add_argument('--n_components', type=int, default=2, help='Number of␣
↪dimensions to reduce to')
    # args = parser.parse_args()
    # reduce_dimensionality(
    #     embeddings_file=args.embeddings_file,
    #     output_file=args.output_file,
    #     n_neighbors=args.n_neighbors,
    #     n_components=args.n_components)
```

```
2024-07-30 20:50:08.987 | INFO     |
__main__:apply_isomap:14 - Applying ISOMAP with

15 neighbors and 2 components
```

UMAP

September 2, 2024

0.0.1 UMAP Sample Calculation

[1]:
```
import numpy as np
from scipy.spatial.distance import pdist, squareform
import umap

# Step 1: Define the input data
points = np.array([
    [-0.70323211, 0.89485216],
    [-0.85881805, 0.4431116],
    [-1.07164979, 1.42083216],
    [-1.52713239, 1.84265852]
])

# Step 2: Calculate pairwise Euclidean distances
pairwise_distances = squareform(pdist(points, metric='euclidean'))
print("Pairwise Distance Matrix:")
print(pairwise_distances)
```

```
Pairwise Distance Matrix:
[[0.         0.47778292 0.6421733  1.25584576]
 [0.47778292 0.         1.00061723 1.55092735]
 [0.6421733  1.00061723 0.         0.62080744]
 [1.25584576 1.55092735 0.62080744 0.        ]]
```

[5]:
```
# Create a copy of the pairwise distance matrix to modify
pairwise_distances_no_self = np.copy(pairwise_distances)

# Set the diagonal to a very large number to effectively ignore self-distances
np.fill_diagonal(pairwise_distances_no_self, np.inf)

# Calculate the local radius (minimum non-self distance for each point)
local_radius = np.min(pairwise_distances_no_self, axis=1)

print("\nLocal Radius (ρ):")
print(local_radius)
```

```
Local Radius (ρ):
```

```
[0.47778292 0.47778292 0.62080744 0.62080744]
```

[6]:
```
# Step 4: Compute Fuzzy Set Membership Strength (μ)
def compute_fuzzy_membership(distances, rho):
    sigma = 1.0  # This is a scaling factor; in practice, this would be␣
 ↪optimized or chosen based on data
    return np.exp(-(distances - rho[:, np.newaxis]) / sigma)

fuzzy_membership = compute_fuzzy_membership(pairwise_distances, local_radius)
print("\nFuzzy Membership Strength (μ):")
print(fuzzy_membership)
```

```
Fuzzy Membership Strength (μ):
[[1.6124954  1.         0.84841075 0.45929488]
 [1.         1.6124954  0.59283788 0.34193165]
 [0.97886077 0.6839915  1.86042962 1.        ]
 [0.52991518 0.3945064  1.         1.86042962]]
```

[7]:
```
# Step 5: Symmetric Fuzzy Set Construction
symmetric_weights = fuzzy_membership + fuzzy_membership.T - fuzzy_membership *␣
 ↪fuzzy_membership.T
print("\nSymmetric Fuzzy Set Weights (w):")
print(symmetric_weights)
```

```
Symmetric Fuzzy Set Weights (w):
[[0.62484938 1.         0.99679552 0.74582273]
 [1.         0.62484938 0.87133331 0.60154383]
 [0.99679552 0.87133331 0.25966087 1.        ]
 [0.74582273 0.60154383 1.         0.25966087]]
```

[8]:
```
# Step 6: Applying UMAP for Dimensionality Reduction
# Initialize UMAP with n_components=1 for 1D projection
umap_reducer = umap.UMAP(n_components=1, random_state=42)

# Fit and transform the data
points_1d = umap_reducer.fit_transform(points)

print("\nReduced 1D Coordinates:")
print(points_1d)
```

```
/Users/vishanthsuresh/anaconda3/lib/python3.11/site-packages/umap/umap_.py:1945:
UserWarning: n_jobs value 1 overridden to 1 by setting random_state. Use no seed
for parallelism.
  warn(f"n_jobs value {self.n_jobs} overridden to 1 by setting random_state. Use
no seed for parallelism.")
/Users/vishanthsuresh/anaconda3/lib/python3.11/site-packages/umap/umap_.py:2437:
UserWarning: n_neighbors is larger than the dataset size; truncating to
```

```
X.shape[0] - 1
  warn(


Reduced 1D Coordinates:
[[32.97621 ]
 [32.166805]
 [30.985752]
 [30.07776 ]]
```

Clustering_Umap_experiments

September 2, 2024

1 Model with UMAP and Kmeans

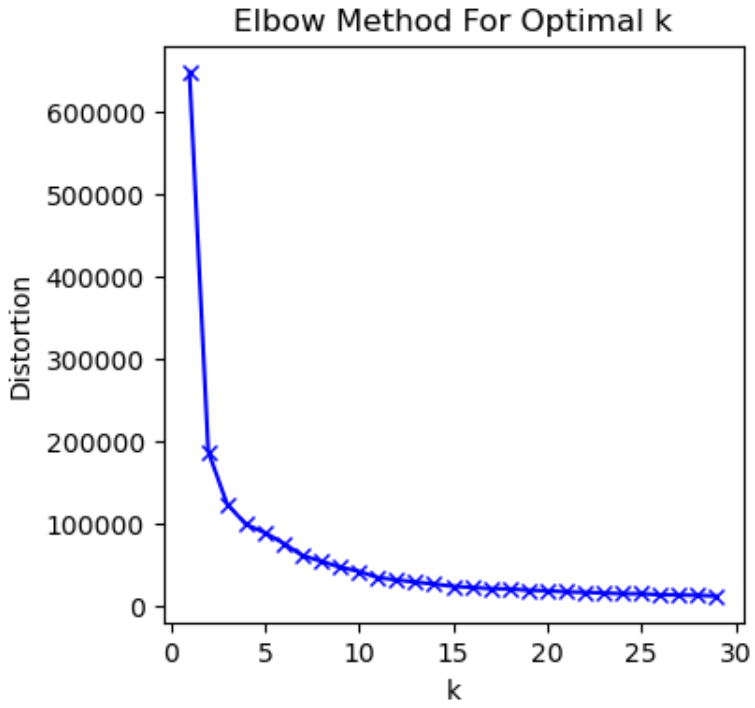
The low dimensional data points are processed for clustering as evaluating high dimensional data seems problematic.

[62]:
```python
import pickle
from argparse import ArgumentParser
from pathlib import Path
from typing import List, Optional, Tuple

import numpy as np
from loguru import logger
from tqdm import tqdm
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from kneed import KneeLocator

def load_embeddings(file: Path) -> Tuple[List[Path], np.ndarray]:
    embeddings = pickle.loads(file.read_bytes())
    files, vectors = zip(*list(embeddings.items()))
    vectors = np.array(vectors)  # Ensure vectors are a NumPy array
    return files, vectors


def make_clusters(vectors: np.ndarray, n_clusters: int) -> np.ndarray:
    logger.info(f"Performing KMeans clustering with {n_clusters} clusters")
    kmeans = KMeans(n_clusters=n_clusters, random_state=42)
    clusters = kmeans.fit_predict(vectors)
    return clusters


def find_optimal_number_of_clusters(data: np.ndarray) -> int:
    logger.info("Finding optimal value for k ...")
    distortions = []
    K = range(1, 30)
    for k in K:
        kmeanModel = KMeans(n_clusters=k)
        kmeanModel.fit(data)
```

```
        distortions.append(kmeanModel.inertia_)

    # Plotting the elbow curve
    plt.figure(figsize=(4,4))
    plt.plot(K, distortions, 'bx-')
    plt.xlabel('k')
    plt.ylabel('Distortion')
    plt.title('Elbow Method For Optimal k')
    plt.show()

    kneedle = KneeLocator(K, distortions, direction='decreasing', curve='convex')
    logger.info(f"Elbow point at k={kneedle.elbow}")
    #return kneedle.elbow if kneedle.elbow else 10# Default to 10 if elbow not␣
 ↪found
    return 4


def compute_clusters(embeddings_file: Path, output_file: Path, clusters:␣
 ↪Optional[int] = None):
    files, vectors = load_embeddings(file=embeddings_file)
    if clusters is None:
        clusters = find_optimal_number_of_clusters(data=vectors)
    clusters = make_clusters(vectors=vectors, n_clusters=clusters)
    cluster_mapping = dict(zip(files, clusters))
    output_file.write_bytes(pickle.dumps(cluster_mapping))


def set_paths_and_run(embeddings_file: str, output_file: str, clusters:␣
 ↪Optional[int] = None):
    compute_clusters(
        embeddings_file=Path(embeddings_file),
        output_file=Path(output_file),
        clusters=clusters
    )


if __name__ == "__main__":
    # Example programmatic usage
    embeddings_file = "/Users/vishanthsuresh/Downloads/Data Science/Dissertation/
 ↪PhIDDLI-main/data/reduced_UMAP.pkl"
    output_file = "/Users/vishanthsuresh/Downloads/Data Science/Dissertation/
 ↪PhIDDLI-main/data/rKMeans_clusters_new.pkl"
    clusters = None  # Or specify an integer for a fixed number of clusters

    set_paths_and_run(embeddings_file, output_file, clusters)
```

```
    # Uncommment the following lines if you want to use command-line arguments␣
↪instead
    # parser = ArgumentParser()
    # parser.add_argument('embeddings_file', type=Path)
    # parser.add_argument('--clusters', type=lambda x: int(x) if x else None,␣
↪default=None, required=False)
    # parser.add_argument('--output-file', type=Path, required=True)
    # args = parser.parse_args()
    # compute_clusters(
    #     embeddings_file=args.embeddings_file,
    #     output_file=args.output_file,
    #     clusters=args.clusters)
```

```
2024-08-09 01:42:04.434 | INFO     |
__main__:find_optimal_number_of_clusters:28 -
Finding optimal value for k ...
```




```
2024-08-09 01:42:05.055 | INFO     |
__main__:find_optimal_number_of_clusters:45 -
Elbow point at k=4
2024-08-09 01:42:05.056 | INFO     |
__main__:make_clusters:21 - Performing KMeans

clustering with 4 clusters
```

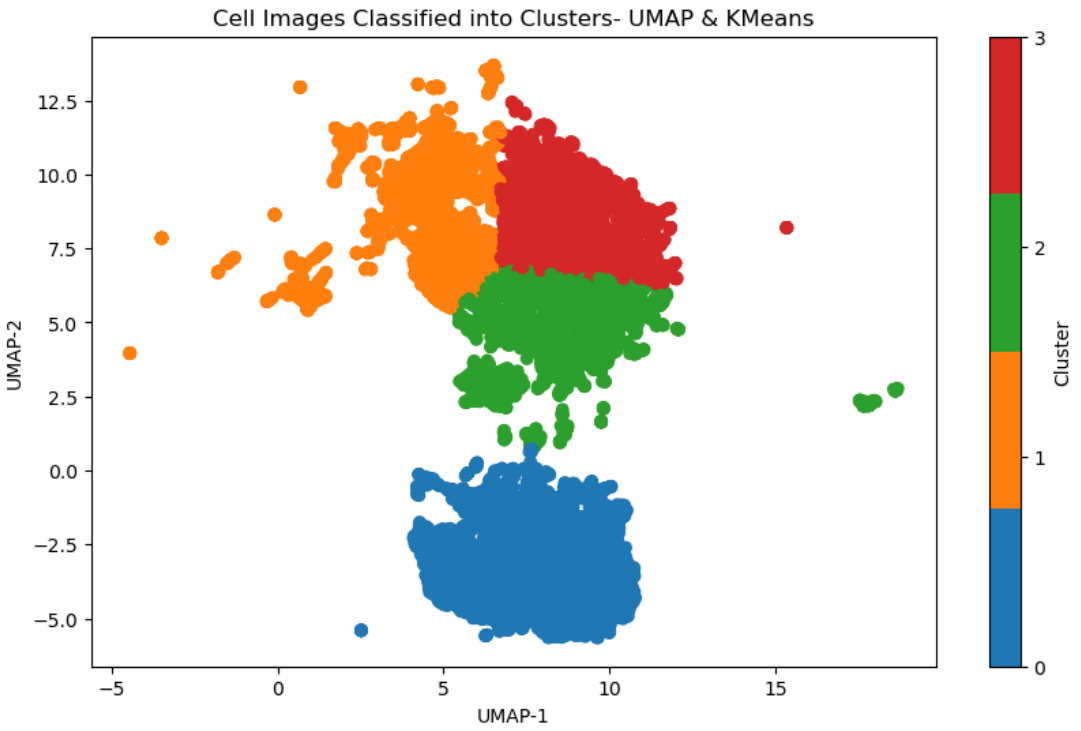
As we can see here that the approach for finding the optimal k value has been changed. In the old PhiDDLI approach, the each k value is tested after getting increamented by 5, whereas here k value is tested by increamenting one by one.

[63]:
```
import pickle
from argparse import ArgumentParser
from collections import defaultdict
from pathlib import Path
from typing import List, Tuple

import pandas as pd

def load_file_contents(file: Path) -> Tuple[List[Path], List]:
    return pickle.loads(file.read_bytes())

def export_csv(clusters_file: Path, points_file: Path, output_file: Path):
    clusters_data = load_file_contents(clusters_file)
    points_data = load_file_contents(points_file)

    data = defaultdict(dict)
    for key, cluster in clusters_data.items():
        data[key]["cluster"] = cluster
        point = points_data[key]
        data[key]["point_x"], data[key]["point_y"] = point

    df: pd.DataFrame = pd.DataFrame.from_dict(data, orient="index") \
        .rename_axis('cell_location') \
        .reset_index() \
        .assign(cell_location=lambda df: df.cell_location.apply(Path)) \
        .assign(parent_image=lambda df: df.cell_location.apply(lambda path: path.
↪parent.name))

    df.to_csv(output_file, index=False)

def set_paths_and_run(clusters_file: str, points_file: str, output_file: str):
    export_csv(
        clusters_file=Path(clusters_file),
        points_file=Path(points_file),
        output_file=Path(output_file)
    )

if __name__ == "__main__":
    # Example programmatic usage
    clusters_file = "/Users/vishanthsuresh/Downloads/Data Science/Dissertation/
↪PhIDDLI-main/data/rKMeans_clusters_new.pkl"
    points_file = "/Users/vishanthsuresh/Downloads/Data Science/Dissertation/
↪PhIDDLI-main/data/reduced_UMAP.pkl"
```

```
    output_file = "/Users/vishanthsuresh/Downloads/Data Science/Dissertation/
↪PhIDDLI-main/data/routput_UMAP_Kmeans_new.csv"

    set_paths_and_run(clusters_file, points_file, output_file)

    # Uncomment the following lines if you want to use command-line arguments␣
↪instead
    # parser = ArgumentParser()
    # parser.add_argument('--clusters', type=Path, required=True)
    # parser.add_argument('--points', type=Path, required=True)
    # parser.add_argument('--output-file', type=Path, required=True)
    # args = parser.parse_args()
    # export_csv(
    #     clusters_file=args.clusters,
    #     points_file=args.points,
    #     output_file=args.output_file)
```

[4]:
```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors


data= pd.read_csv("/Users/vishanthsuresh/Downloads/Data Science/Dissertation/
↪PhIDDLI-main/data/routput_UMAP_Kmeans_new.csv")
# Define a set of more contrasting and visually distinct colors
contrasting_colors = [
    '#1f77b4', '#ff7f0e', '#2ca02c','#d62728'
]

# Create a colormap from the list of contrasting colors
custom_cmap_contrasting = mcolors.ListedColormap(contrasting_colors[:15])

# Plotting point_x and point_y with clusters in different colors using the␣
↪custom colormap
plt.figure(figsize=(10, 6))

# Using a scatter plot to visualize the clusters
scatter = plt.scatter(data['point_x'], data['point_y'], c=data['cluster'],␣
↪cmap=custom_cmap_contrasting)

# Adding a color bar to indicate the cluster colors
cbar = plt.colorbar(scatter, label='Cluster', ticks=range(15))
cbar.set_ticklabels(range(15))

# Adding titles and labels
plt.title('Cell Images Classified into Clusters- UMAP & KMeans')
plt.xlabel('UMAP-1')
```

```
plt.ylabel('UMAP-2')

# Display the plot
plt.show()
```



[65]:
```
import pandas as pd
from sklearn.metrics import silhouette_score, calinski_harabasz_score,␣
 ↪davies_bouldin_score

# Load the CSV file
file_path = '/Users/vishanthsuresh/Downloads/Data Science/Dissertation/
 ↪PhIDDLI-main/data/routput_UMAP_Kmeans_new.csv'
data = pd.read_csv(file_path)

# Extracting the relevant data for clustering validation
vectors = data[['point_x', 'point_y']].values
labels = data['cluster'].values

# Calculate the validation metrics
silhouette_avg = silhouette_score(vectors, labels)
calinski_harabasz_avg = calinski_harabasz_score(vectors, labels)
davies_bouldin_avg = davies_bouldin_score(vectors, labels)
```

```
print(f"Silhouette Score: {silhouette_avg}")
print(f"Calinski-Harabasz Index: {calinski_harabasz_avg}")
print(f"Davies-Bouldin Index: {davies_bouldin_avg}")
```

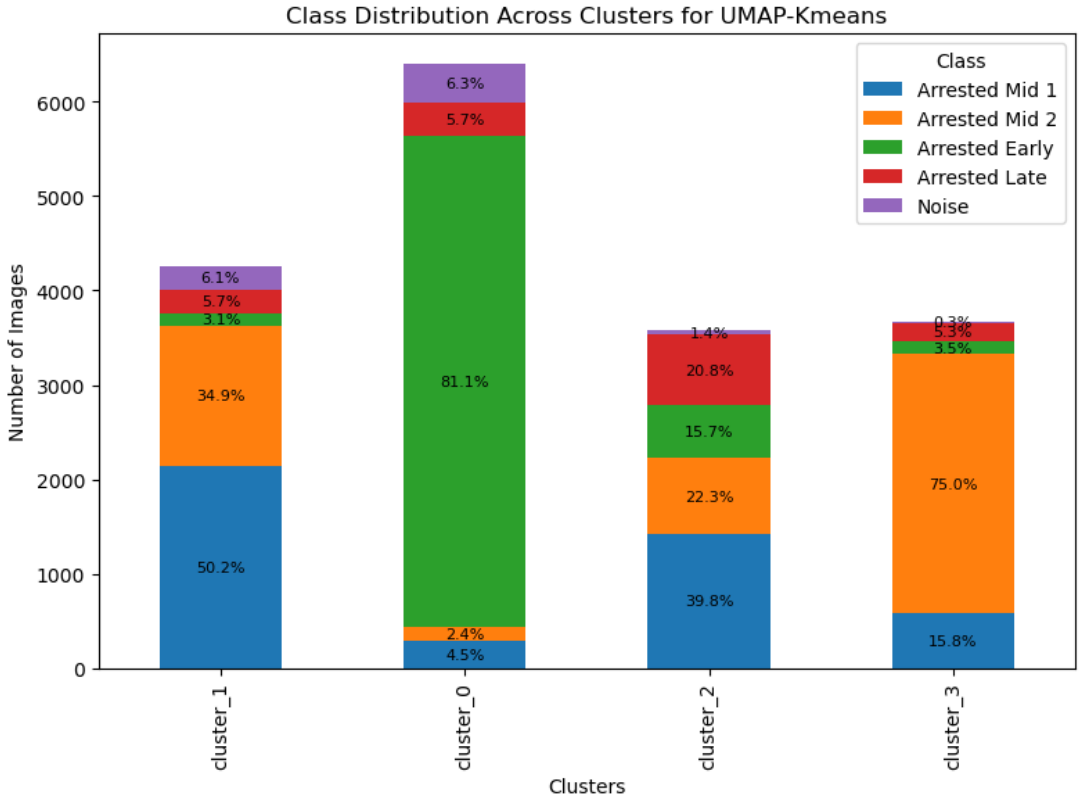
```
Silhouette Score: 0.46425583977749024
Calinski-Harabasz Index: 33310.15230476035
Davies-Bouldin Index: 0.8345935422076254
```

[3]:
```
import os
import torch
import torch.nn as nn
from torchvision import transforms
from PIL import Image
import pandas as pd
import matplotlib.pyplot as plt

# Define CNN Model with three hidden layers (same as before)
class CustomCNN(nn.Module):
    def __init__(self):
        super(CustomCNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.maxpool1 = nn.MaxPool2d(kernel_size=2)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, padding=1)
        self.conv4 = nn.Conv2d(128, 128, kernel_size=3, padding=1)
        self.maxpool2 = nn.MaxPool2d(kernel_size=2)
        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(128 * 8 * 8, 512)
        self.fc2 = nn.Linear(512, 256)  # Third hidden layer
        self.fc3 = nn.Linear(256, 128)  # Fourth hidden layer
        self.fc4 = nn.Linear(128, 30)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.maxpool1(self.relu(self.conv1(x)))
        x = self.maxpool1(self.relu(self.conv2(x)))
        x = self.relu(self.conv3(x))
        x = self.maxpool2(self.relu(self.conv4(x)))
        x = self.flatten(x)
        x = self.relu(self.fc1(x))
        x = self.relu(self.fc2(x))  # Third hidden layer
        x = self.relu(self.fc3(x))  # Fourth hidden layer
        x = self.fc4(x)
        return x

# Load the checkpoint
```

```
checkpoint = torch.load('/Users/vishanthsuresh/Downloads/Data Science/
 ↪Dissertation/CNN Model/final_model_checkpoint.pth')
model = CustomCNN()
model.load_state_dict(checkpoint['model_state_dict'])
model.eval()

# Define the transformation for the images
transform = transforms.Compose([
    transforms.Resize((64, 64)),
    transforms.ToTensor(),
])

# Define a dictionary to map class indices to class names
class_names = {
    0: 'Arrested Mid 1',
    1: 'Arrested Mid 2',
    2: 'Arrested Early',
    3: 'Arrested Late',
    4: 'Noise'
    # Add other class mappings as necessary
}

# Function to predict image
def predict_image(image_path, model, transform):
    image = Image.open(image_path)
    image = transform(image)
    image = image.unsqueeze(0)  # Add batch dimension
    with torch.no_grad():
        output = model(image)
    _, predicted = torch.max(output, 1)
    return class_names[predicted.item()]

# Directory containing all the clusters
base_folder = '/Users/vishanthsuresh/Downloads/Data Science/Dissertation/
 ↪PhIDDLI-main/data/clusters_umap_kmeans'
# Initialize an empty DataFrame with columns for each class
all_class_counts = pd.DataFrame(columns=class_names.values())
cluster_names = []

# Iterate over all subfolders and predict images
for folder_name in os.listdir(base_folder):
    folder_path = os.path.join(base_folder, folder_name)
    if os.path.isdir(folder_path):  # Ensure it's a directory
        results = []
        for image_name in os.listdir(folder_path):
            if image_name.endswith(('.png', '.jpg', '.jpeg')):  # Add other␣
 ↪image extensions if needed
```

```
                image_path = os.path.join(folder_path, image_name)
                predicted_class = predict_image(image_path, model, transform)
                results.append({'Image': image_name, 'Predicted_Class':␣
↪predicted_class})

        # Save results to CSV
        results_df = pd.DataFrame(results)
        results_csv_path = os.path.join(folder_path, f'{folder_name}_predictions.
↪csv')
        results_df.to_csv(results_csv_path, index=False)
        print(f"Results saved to {results_csv_path}")

        # Store class counts for the current cluster
        class_counts = results_df['Predicted_Class'].value_counts()
        # Add missing classes with 0 count
        class_counts = class_counts.reindex(all_class_counts.columns,␣
↪fill_value=0)
        all_class_counts = pd.concat([all_class_counts, class_counts.to_frame().
↪T], ignore_index=True)
        cluster_names.append(folder_name)

# Set index to cluster names
all_class_counts.index = cluster_names

# Plot stacked bar chart with percentage annotations
fig, ax = plt.subplots(figsize=(8, 6))
bars = all_class_counts.plot(kind='bar', stacked=True, ax=ax)

# Calculate percentages and annotate
for i, cluster in enumerate(all_class_counts.index):
    total = all_class_counts.loc[cluster].sum()
    cumulative_sum = 0
    for j, class_name in enumerate(all_class_counts.columns):
        class_count = all_class_counts.loc[cluster, class_name]
        cumulative_sum += class_count
        if class_count > 0:
            percentage = (class_count / total) * 100
            ax.text(i, cumulative_sum - class_count / 2, f'{percentage:.1f}%',␣
↪ha='center', va='center', fontsize=8)

plt.title('Class Distribution Across Clusters for UMAP-Kmeans')
plt.xlabel('Clusters')
plt.ylabel('Number of Images')
plt.legend(title='Class')
plt.tight_layout()
plt.show()
```

```
Results saved to /Users/vishanthsuresh/Downloads/Data
Science/Dissertation/PhIDDLI-
main/data/clusters_umap_kmeans/cluster_1/cluster_1_predictions.csv
Results saved to /Users/vishanthsuresh/Downloads/Data
Science/Dissertation/PhIDDLI-
main/data/clusters_umap_kmeans/cluster_0/cluster_0_predictions.csv
Results saved to /Users/vishanthsuresh/Downloads/Data
Science/Dissertation/PhIDDLI-
main/data/clusters_umap_kmeans/cluster_2/cluster_2_predictions.csv
Results saved to /Users/vishanthsuresh/Downloads/Data
Science/Dissertation/PhIDDLI-
main/data/clusters_umap_kmeans/cluster_3/cluster_3_predictions.csv
```



The images has been clustered and segregated into separate folders for analysis.

2 UMAP and Hierarchical Clustering

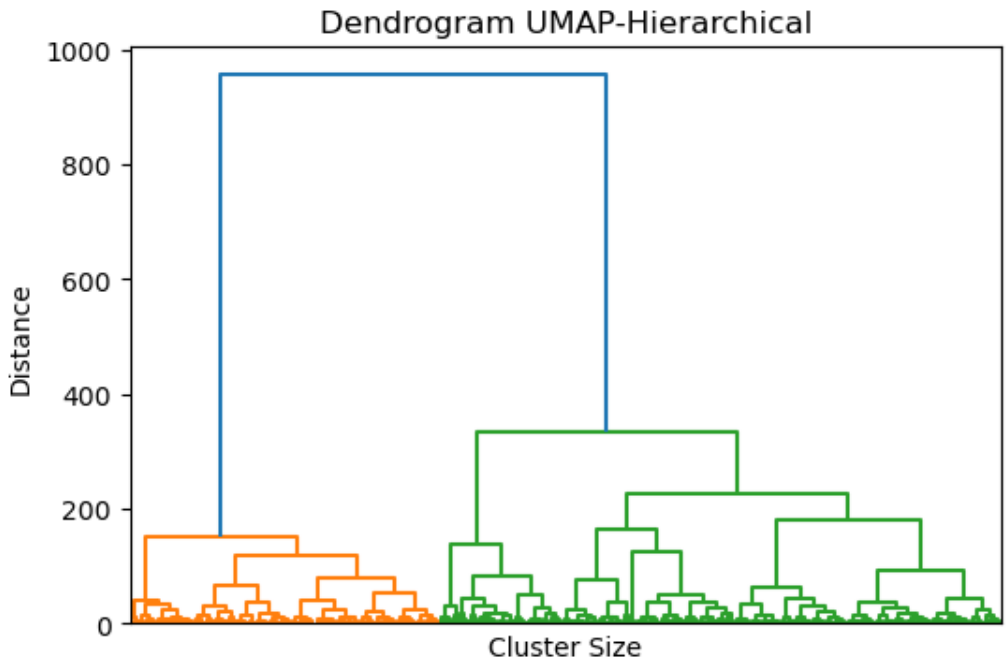
[37]:
```
import pickle
from pathlib import Path
from typing import List, Tuple
```

```
import numpy as np
from loguru import logger
from scipy.cluster.hierarchy import linkage, dendrogram
import matplotlib.pyplot as plt

def load_embeddings(file: Path) -> Tuple[List[Path], np.ndarray]:
    with file.open('rb') as f:
        embeddings = pickle.load(f)
    files, vectors = zip(*list(embeddings.items()))
    vectors = np.array(vectors)  # Ensure vectors are a NumPy array
    return files, vectors

def plot_dendrogram(vectors: np.ndarray, output_file: Path):
    Z = linkage(vectors, method='ward')
    plt.figure(figsize=(10, 7))
    dendrogram(Z)
    plt.title('Dendrogram')
    plt.xlabel('Data Points')
    plt.ylabel('Distance')
    plt.savefig(output_file)
    plt.close()
    logger.info(f"Dendrogram plot saved to {output_file}")

def compute_dendrogram(embeddings_file: Path, dendrogram_file: Path):
    files, vectors = load_embeddings(file=embeddings_file)
    plot_dendrogram(vectors, dendrogram_file)

def set_paths_and_run(embeddings_file: str, dendrogram_file: str):
    compute_dendrogram(
        embeddings_file=Path(embeddings_file),
        dendrogram_file=Path(dendrogram_file)
    )

if __name__ == "__main__":
    # Example programmatic usage
    embeddings_file = "/Users/vishanthsuresh/Downloads/Data Science/Dissertation/
 ↪PhIDDLI-main/data/reduced_UMAP.pkl"
    dendrogram_file = "/Users/vishanthsuresh/Downloads/Data Science/Dissertation/
 ↪PhIDDLI-main/data/dendrogram.png"

    set_paths_and_run(embeddings_file, dendrogram_file)

    # Uncomment the following lines if you want to use command-line arguments␣
 ↪instead
    # parser = ArgumentParser()
    # parser.add_argument('embeddings_file', type=Path)
    # parser.add_argument('--dendrogram-file', type=Path, required=True)
```

```
    # args = parser.parse_args()
    # compute_dendrogram(
    #     embeddings_file=args.embeddings_file,
    #     dendrogram_file=args.dendrogram_file)
```

```
2024-07-13 16:04:54.269 | INFO     |
__main__:plot_dendrogram:26 - Dendrogram plot
saved to /Users/vishanthsuresh/Downloads/Data Science/Dissertation/PhIDDLI-
main/data/dendrogram.png
```

[8]:
```
import pickle
from scipy.cluster.hierarchy import linkage, dendrogram
import matplotlib.pyplot as plt
import numpy as np

# Define the path to the embeddings file
embeddings_file_path = '/Users/vishanthsuresh/Downloads/Data Science/
 ↪Dissertation/PhIDDLI-main/data/reduced_UMAP.pkl'

# Load the embeddings from the pickle file
with open(embeddings_file_path, 'rb') as file:
    embeddings = pickle.load(file)

# Extract the vectors from the loaded data
files, vectors = zip(*list(embeddings.items()))
vectors = np.array(vectors)  # Ensure vectors are a NumPy array

# Compute the linkage matrix
Z = linkage(vectors, method='ward')

# Plot the dendrogram without data point markers
plt.figure(figsize=(6, 4))
plt.title("Dendrogram UMAP-Hierarchical")
dendrogram(Z, leaf_rotation=90, leaf_font_size=10, no_labels=True)
plt.xlabel('Cluster Size')
plt.ylabel('Distance')
plt.show()
```



2.0.1 Choosing the optimal number of Clusters

From Dendogram the number of optimal clusters is chosen as 6 as the thershold line at the lowest point of the longest vertical line passes through six lines.

[39]:
```
import pickle
from argparse import ArgumentParser
from pathlib import Path
from typing import List, Optional, Tuple

import numpy as np
from loguru import logger
from tqdm import tqdm
from scipy.cluster.hierarchy import linkage, fcluster
import hdbscan

def load_embeddings(file: Path) -> Tuple[List[Path], np.ndarray]:
    with file.open('rb') as f:
        embeddings = pickle.load(f)
    files, vectors = zip(*list(embeddings.items()))
    vectors = np.array(vectors)  # Ensure vectors are a NumPy array
    return files, vectors

def make_clusters(vectors: np.ndarray, n_clusters: int) -> np.ndarray:
    Z = linkage(vectors, method='ward')
```

```
    logger.info(f"Performing hierarchical clustering with {n_clusters} clusters")
    return fcluster(Z, n_clusters, criterion='maxclust')

def find_optimal_number_of_clusters(data: np.ndarray) -> int:
    #logger.info("Finding optimal value for k using HDBSCAN ...")
    #clusterer = hdbscan.HDBSCAN(min_cluster_size=100)
    #clusterer.fit(data)

    #labels = clusterer.labels_
    #num_clusters = len(set(labels)) - (1 if -1 in labels else 0)
    #logger.info(f"Optimal number of clusters determined by HDBSCAN:␣
↪{num_clusters}")
    #return num_clusters
    return 6

def compute_clusters(embeddings_file: Path, output_file: Path, clusters:␣
↪Optional[int] = None):
    files, vectors = load_embeddings(file=embeddings_file)
    if clusters is None:
        clusters = find_optimal_number_of_clusters(data=vectors)
    cluster_labels = make_clusters(vectors=vectors, n_clusters=clusters)
    cluster_mapping = dict(zip(files, cluster_labels))
    with output_file.open('wb') as f:
        pickle.dump(cluster_mapping, f)

def set_paths_and_run(embeddings_file: str, output_file: str, clusters:␣
↪Optional[int] = None):
    compute_clusters(
        embeddings_file=Path(embeddings_file),
        output_file=Path(output_file),
        clusters=clusters
    )

if __name__ == "__main__":
    # Example programmatic usage
    embeddings_file = "/Users/vishanthsuresh/Downloads/Data Science/Dissertation/
↪PhIDDLI-main/data/reduced_UMAP.pkl"
    output_file = "/Users/vishanthsuresh/Downloads/Data Science/Dissertation/
↪PhIDDLI-main/data/rHierarichal_clusters_UMAP.pkl"
    clusters = None  # Or specify an integer for a fixed number of clusters

    set_paths_and_run(embeddings_file, output_file, clusters)

    # Uncomment the following lines if you want to use command-line arguments␣
↪instead
    # parser = ArgumentParser()
```

```
    # parser.add_argument('embeddings_file', type=Path)
    # parser.add_argument('--clusters', type=lambda x: int(x) if x else None,␣
↪default=None, required=False)
    # parser.add_argument('--output-file', type=Path, required=True)
    # args = parser.parse_args()
    # compute_clusters(
    #     embeddings_file=args.embeddings_file,
    #     output_file=args.output_file,
    #     clusters=args.clusters)
```

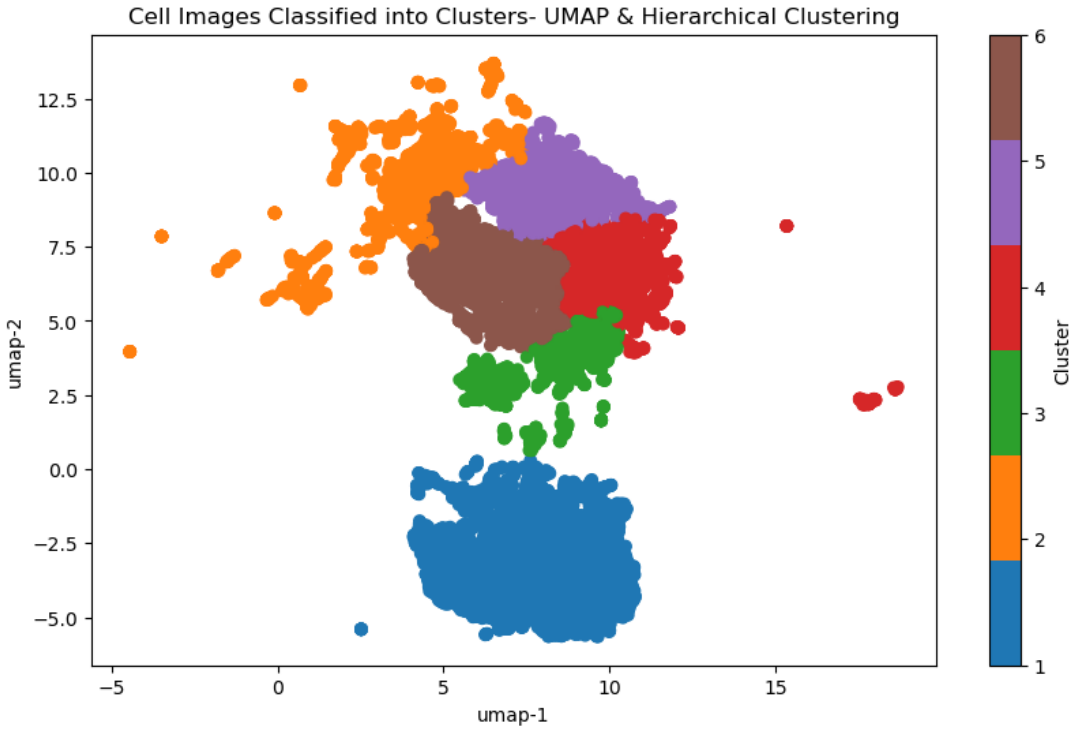
```
2024-08-07 23:17:57.305 | INFO     |
__main__:make_clusters:21 - Performing

hierarchical clustering with 6 clusters
```

[52]:
```
import pickle
from argparse import ArgumentParser
from collections import defaultdict
from pathlib import Path
from typing import List, Tuple

import pandas as pd

def load_file_contents(file: Path) -> Tuple[List[Path], List]:
    return pickle.loads(file.read_bytes())

def export_csv(clusters_file: Path, points_file: Path, output_file: Path):
    clusters_data = load_file_contents(clusters_file)
    points_data = load_file_contents(points_file)

    data = defaultdict(dict)
    for key, cluster in clusters_data.items():
        data[key]["cluster"] = cluster
        point = points_data[key]
        data[key]["point_x"], data[key]["point_y"] = point

    df: pd.DataFrame = pd.DataFrame.from_dict(data, orient="index") \
        .rename_axis('cell_location') \
        .reset_index() \
        .assign(cell_location=lambda df: df.cell_location.apply(Path)) \
        .assign(parent_image=lambda df: df.cell_location.apply(lambda path: path.
↪parent.name))

    df.to_csv(output_file, index=False)

def set_paths_and_run(clusters_file: str, points_file: str, output_file: str):
    export_csv(
        clusters_file=Path(clusters_file),
```

```
        points_file=Path(points_file),
        output_file=Path(output_file)
    )

if __name__ == "__main__":
    # Example programmatic usage
    clusters_file = "/Users/vishanthsuresh/Downloads/Data Science/Dissertation/
 ↪PhIDDLI-main/data/rHierarichal_clusters_UMAP.pkl"
    points_file = "/Users/vishanthsuresh/Downloads/Data Science/Dissertation/
 ↪PhIDDLI-main/data/reduced_UMAP.pkl"
    output_file = "/Users/vishanthsuresh/Downloads/Data Science/Dissertation/
 ↪PhIDDLI-main/data/output_hierarchical_umap.csv"

    set_paths_and_run(clusters_file, points_file, output_file)

    # Uncomment the following lines if you want to use command-line arguments␣
 ↪instead
    # parser = ArgumentParser()
    # parser.add_argument('--clusters', type=Path, required=True)
    # parser.add_argument('--points', type=Path, required=True)
    # parser.add_argument('--output-file', type=Path, required=True)
    # args = parser.parse_args()
    # export_csv(
    #     clusters_file=args.clusters,
    #     points_file=args.points,
    #     output_file=args.output_file)
```

[7]:
```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors


data= pd.read_csv("/Users/vishanthsuresh/Downloads/Data Science/Dissertation/
 ↪PhIDDLI-main/data/output_hierarchical_umap.csv")
# Define a set of more contrasting and visually distinct colors
contrasting_colors = [
    '#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd',
    '#8c564b'
]

# Create a colormap from the list of contrasting colors
custom_cmap_contrasting = mcolors.ListedColormap(contrasting_colors[:15])

# Plotting point_x and point_y with clusters in different colors using the␣
 ↪custom colormap
plt.figure(figsize=(10, 6))
```

```
# Using a scatter plot to visualize the clusters
scatter = plt.scatter(data['point_x'], data['point_y'], c=data['cluster'],␣
↪cmap=custom_cmap_contrasting)

# Adding a color bar to indicate the cluster colors
cbar = plt.colorbar(scatter, label='Cluster', ticks=range(15))
cbar.set_ticklabels(range(15))

# Adding titles and labels
plt.title('Cell Images Classified into Clusters- UMAP & Hierarchical Clustering')
plt.xlabel('umap-1')
plt.ylabel('umap-2')

# Display the plot
plt.show()
```



[55]:
```
import pandas as pd
from sklearn.metrics import silhouette_score, calinski_harabasz_score,␣
↪davies_bouldin_score

# Load the CSV file
```

```
file_path = '/Users/vishanthsuresh/Downloads/Data Science/Dissertation/
 ↪PhIDDLI-main/data/output_hierarchical_umap.csv'
data = pd.read_csv(file_path)

# Extracting the relevant data for clustering validation
vectors = data[['point_x', 'point_y']].values
labels = data['cluster'].values

# Calculate the validation metrics
silhouette_avg = silhouette_score(vectors, labels)
calinski_harabasz_avg = calinski_harabasz_score(vectors, labels)
davies_bouldin_avg = davies_bouldin_score(vectors, labels)

print(f"Silhouette Score: {silhouette_avg}")
print(f"Calinski-Harabasz Index: {calinski_harabasz_avg}")
print(f"Davies-Bouldin Index: {davies_bouldin_avg}")
```

```
Silhouette Score: 0.41658099104234314
Calinski-Harabasz Index: 25387.171079316387
Davies-Bouldin Index: 0.8338531148866153
```

[5]:
```
import os
import shutil
import pandas as pd

# Load the data
file_path = '/Users/vishanthsuresh/Downloads/Data Science/Dissertation/
 ↪PhIDDLI-main/data/output_hierarchical_umap.csv'
data = pd.read_csv(file_path)

# Base directory where images are stored
base_image_dir = '/Users/vishanthsuresh/Downloads/Data Science/Dissertation/
 ↪PhIDDLI-main/data/extracted_cells'

# Base directory to store segregated clusters
output_dir = '/Users/vishanthsuresh/Downloads/Data Science/Dissertation/
 ↪PhIDDLI-main/data/clusters_umap_hierarchical'

# Function to get all image paths from subdirectories
def get_all_image_paths(base_dir):
    image_paths = []
    for root, _, files in os.walk(base_dir):
        for file in files:
            if file.endswith(('.jpg', '.jpeg', '.png', '.tif', '.tiff')):  # Add␣
 ↪other image extensions if needed
                image_paths.append(os.path.join(root, file))
    return image_paths
```

```
# Get all image paths from the base image directory
all_image_paths = get_all_image_paths(base_image_dir)

# Create a dictionary to map relative cell locations to their paths
image_path_dict = {os.path.relpath(path, base_image_dir): path for path in␣
↪all_image_paths}

# Create directories for each cluster
unique_clusters = data['cluster'].unique()
for cluster in unique_clusters:
    cluster_dir = os.path.join(output_dir, f'cluster_{cluster}')
    if not os.path.exists(cluster_dir):
        os.makedirs(cluster_dir)

# Move files to corresponding cluster directories
for index, row in data.iterrows():
    cluster = row['cluster']
    relative_path = os.path.relpath(row['cell_location'], base_image_dir)
    source_file_path = image_path_dict.get(relative_path)

    if source_file_path and os.path.exists(source_file_path):
        # Create a unique destination file path
        unique_filename = f"{os.path.splitext(relative_path.replace('/',␣
↪'_'))[0]}_{index}{os.path.splitext(relative_path)[1]}"
        dest_file_path = os.path.join(output_dir, f'cluster_{cluster}',␣
↪unique_filename)

        # Ensure the directory exists
        dest_dir = os.path.dirname(dest_file_path)
        if not os.path.exists(dest_dir):
            os.makedirs(dest_dir)

        shutil.copy(source_file_path, dest_file_path)
    else:
        print(f"File not found for: {relative_path}")

print("Images segregated into cluster folders successfully.")
```

```
Images segregated into cluster folders successfully.
```

[6]:
```
import os
import torch
import torch.nn as nn
from torchvision import transforms
from PIL import Image
import pandas as pd
```

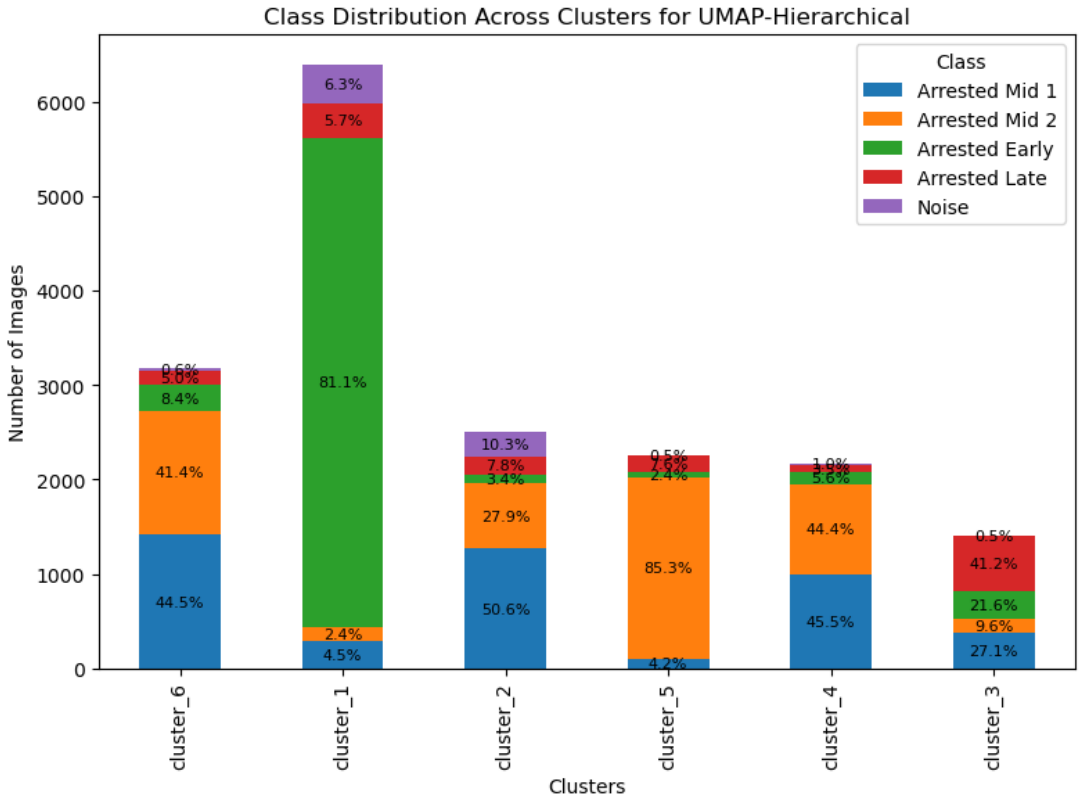
```
import matplotlib.pyplot as plt

# Define CNN Model with three hidden layers (same as before)
class CustomCNN(nn.Module):
    def __init__(self):
        super(CustomCNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.maxpool1 = nn.MaxPool2d(kernel_size=2)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, padding=1)
        self.conv4 = nn.Conv2d(128, 128, kernel_size=3, padding=1)
        self.maxpool2 = nn.MaxPool2d(kernel_size=2)
        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(128 * 8 * 8, 512)
        self.fc2 = nn.Linear(512, 256)  # Third hidden layer
        self.fc3 = nn.Linear(256, 128)  # Fourth hidden layer
        self.fc4 = nn.Linear(128, 30)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.maxpool1(self.relu(self.conv1(x)))
        x = self.maxpool1(self.relu(self.conv2(x)))
        x = self.relu(self.conv3(x))
        x = self.maxpool2(self.relu(self.conv4(x)))
        x = self.flatten(x)
        x = self.relu(self.fc1(x))
        x = self.relu(self.fc2(x))  # Third hidden layer
        x = self.relu(self.fc3(x))  # Fourth hidden layer
        x = self.fc4(x)
        return x

# Load the checkpoint
checkpoint = torch.load('/Users/vishanthsuresh/Downloads/Data Science/
 ↪Dissertation/CNN Model/final_model_checkpoint.pth')
model = CustomCNN()
model.load_state_dict(checkpoint['model_state_dict'])
model.eval()

# Define the transformation for the images
transform = transforms.Compose([
    transforms.Resize((64, 64)),
    transforms.ToTensor(),
])

# Define a dictionary to map class indices to class names
class_names = {
    0: 'Arrested Mid 1',
```

```
    1: 'Arrested Mid 2',
    2: 'Arrested Early',
    3: 'Arrested Late',
    4: 'Noise'
    # Add other class mappings as necessary
}

# Function to predict image
def predict_image(image_path, model, transform):
    image = Image.open(image_path)
    image = transform(image)
    image = image.unsqueeze(0)  # Add batch dimension
    with torch.no_grad():
        output = model(image)
    _, predicted = torch.max(output, 1)
    return class_names[predicted.item()]

# Directory containing all the clusters
base_folder = '/Users/vishanthsuresh/Downloads/Data Science/Dissertation/
 ↪PhIDDLI-main/data/clusters_umap_hierarchical'
# Initialize an empty DataFrame with columns for each class
all_class_counts = pd.DataFrame(columns=class_names.values())
cluster_names = []

# Iterate over all subfolders and predict images
for folder_name in os.listdir(base_folder):
    folder_path = os.path.join(base_folder, folder_name)
    if os.path.isdir(folder_path):  # Ensure it's a directory
        results = []
        for image_name in os.listdir(folder_path):
            if image_name.endswith(('.png', '.jpg', '.jpeg')):  # Add other␣
 ↪image extensions if needed
                image_path = os.path.join(folder_path, image_name)
                predicted_class = predict_image(image_path, model, transform)
                results.append({'Image': image_name, 'Predicted_Class':␣
 ↪predicted_class})

        # Save results to CSV
        results_df = pd.DataFrame(results)
        results_csv_path = os.path.join(folder_path, f'{folder_name}_predictions.
 ↪csv')
        results_df.to_csv(results_csv_path, index=False)
        print(f"Results saved to {results_csv_path}")

        # Store class counts for the current cluster
        class_counts = results_df['Predicted_Class'].value_counts()
        # Add missing classes with 0 count
```

```
        class_counts = class_counts.reindex(all_class_counts.columns,␣
↪fill_value=0)
        all_class_counts = pd.concat([all_class_counts, class_counts.to_frame().
↪T], ignore_index=True)
        cluster_names.append(folder_name)

# Set index to cluster names
all_class_counts.index = cluster_names

# Plot stacked bar chart with percentage annotations
fig, ax = plt.subplots(figsize=(8, 6))
bars = all_class_counts.plot(kind='bar', stacked=True, ax=ax)

# Calculate percentages and annotate
for i, cluster in enumerate(all_class_counts.index):
    total = all_class_counts.loc[cluster].sum()
    cumulative_sum = 0
    for j, class_name in enumerate(all_class_counts.columns):
        class_count = all_class_counts.loc[cluster, class_name]
        cumulative_sum += class_count
        if class_count > 0:
            percentage = (class_count / total) * 100
            ax.text(i, cumulative_sum - class_count / 2, f'{percentage:.1f}%',␣
↪ha='center', va='center', fontsize=8)

plt.title('Class Distribution Across Clusters for UMAP-Hierarchical')
plt.xlabel('Clusters')
plt.ylabel('Number of Images')
plt.legend(title='Class')
plt.tight_layout()
plt.show()
```

```
Results saved to /Users/vishanthsuresh/Downloads/Data
Science/Dissertation/PhIDDLI-
main/data/clusters_umap_hierarchical/cluster_6/cluster_6_predictions.csv
Results saved to /Users/vishanthsuresh/Downloads/Data
Science/Dissertation/PhIDDLI-
main/data/clusters_umap_hierarchical/cluster_1/cluster_1_predictions.csv
Results saved to /Users/vishanthsuresh/Downloads/Data
Science/Dissertation/PhIDDLI-
main/data/clusters_umap_hierarchical/cluster_2/cluster_2_predictions.csv
Results saved to /Users/vishanthsuresh/Downloads/Data
Science/Dissertation/PhIDDLI-
main/data/clusters_umap_hierarchical/cluster_5/cluster_5_predictions.csv
Results saved to /Users/vishanthsuresh/Downloads/Data
Science/Dissertation/PhIDDLI-
main/data/clusters_umap_hierarchical/cluster_4/cluster_4_predictions.csv
```

```
Results saved to /Users/vishanthsuresh/Downloads/Data
Science/Dissertation/PhIDDLI-
main/data/clusters_umap_hierarchical/cluster_3/cluster_3_predictions.csv
```



The images for all the clusters are segregated and stored on the Onedrive. Link has been provided in the email

Semi Supervised Fuzzy Clustering

September 2, 2024

0.0.1 Semi Supervised Fuzzy Clustering

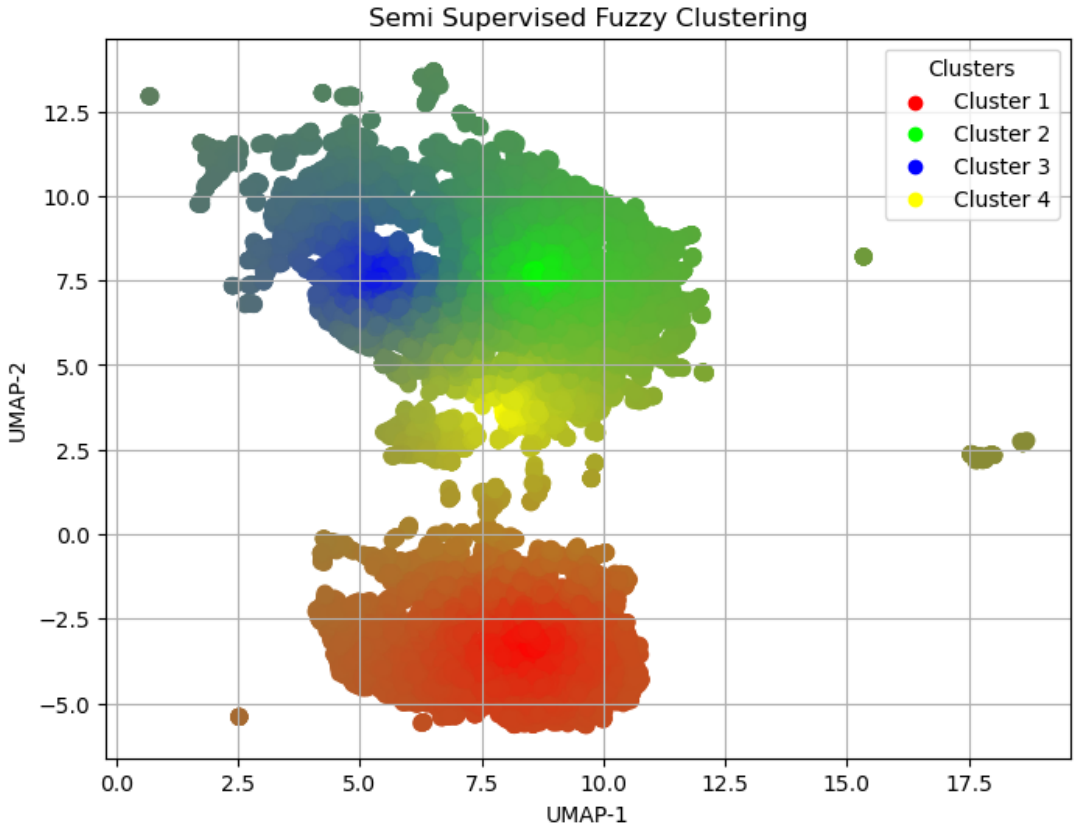
[31]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import normalize
from skfuzzy import cmeans

# Load the UMAP-reduced embeddings
umap_embeddings = pd.read_csv('/Users/vishanthsuresh/Downloads/Data Science/
 ↪Dissertation/PhIDDLI-main/data/UMAP_fuzzy_data_points.csv')
data_points = umap_embeddings[['point_x', 'point_y']].values

# Load the labeled data
labeled_data_df = pd.read_csv('/Users/vishanthsuresh/Downloads/Data Science/
 ↪Dissertation/PhIDDLI-main/data/partial_label_V2.0.csv')  # Update with actual␣
 ↪path to the labeled data CSV
labeled_data_points = labeled_data_df[['point_x', 'point_y']].values
labels = labeled_data_df['class'].values

# Combine data points and labeled data for initial clustering
combined_data = np.vstack([data_points, labeled_data_points])

# Define Fuzzy C-Means parameters
n_clusters = len(np.unique(labels))  # Number of clusters (same as number of␣
 ↪unique classes)
fuzziness = 3.0  # Fuzzification parameter
max_iter = 100
error = 0.005
alpha = 0.5  # Weight for partial supervision

# Perform initial Fuzzy C-Means clustering
cntr, u, _, _, _, _, _ = cmeans(combined_data.T, n_clusters, fuzziness,␣
 ↪error=error, maxiter=max_iter)

# Update membership matrix u with partial supervision
labeled_count = len(labeled_data_points)
full_count = len(combined_data)
```

```
# Create a mapping from class names to cluster indices
class_names = np.unique(labels)
class_to_cluster_index = {class_name: idx for idx, class_name in␣
↪enumerate(class_names)}

for j in range(labeled_count):
    class_name = labels[j]
    cluster_index = class_to_cluster_index[class_name]
    u[cluster_index, -labeled_count + j] = 1

u = normalize(u, norm='l1', axis=0)

# Extract the fuzzy membership values for the original data points
membership_values = u[:, :-labeled_count].T

# Add membership values for each cluster to the DataFrame
for idx in range(n_clusters):
    umap_embeddings[f'membership_{idx + 1}'] = membership_values[:, idx]

# Add cluster labels to the DataFrame (label clusters from 1 to 5)
umap_embeddings['cluster'] = np.argmax(membership_values, axis=1) + 1

# Save the results to the same CSV file
umap_embeddings.to_csv('/Users/vishanthsuresh/Downloads/Data Science/
↪Dissertation/PhIDDLI-main/data/UMAP_fuzzy_data_points.csv', index=False)

print('Clustering completed and results saved.')
```

```
Clustering completed and results saved.
```

[48]:
```
import matplotlib.pyplot as plt
import numpy as np
import mplcursors

data= pd.read_csv("/Users/vishanthsuresh/Downloads/Data Science/Dissertation/
↪PhIDDLI-main/data/UMAP_fuzzy_data_points.csv")
# Assuming 'data' is already loaded as shown in your code
# Extract relevant data
x = data['point_x']
y = data['point_y']
memberships = data[['membership_1', 'membership_2', 'membership_3',␣
↪'membership_4']]

# Normalize membership values to get a color for each point
colors = memberships.values.dot(np.array([[1, 0, 0], [0, 1, 0], [0, 0, 1], [1,␣
↪1, 0]]))
colors = colors / np.max(colors, axis=0)  # Normalize colors to [0, 1]
```

```
# Create the scatter plot
plt.figure(figsize=(8, 6))
scatter = plt.scatter(x, y, c=colors, s=50)

# Manually add legend for each cluster
for i, color in enumerate([[1, 0, 0], [0, 1, 0], [0, 0, 1], [1, 1, 0]]):
    plt.scatter([], [], c=[color], label=f'Cluster {i+1}')

plt.title('Semi Supervised Fuzzy Clustering')
plt.xlabel('UMAP-1')
plt.ylabel('UMAP-2')
plt.legend(title="Clusters")
plt.grid(True)

# Add interactive cursor
cursor = mplcursors.cursor(scatter, hover=True)

# Display the coordinates when a point is clicked
@cursor.connect("add")
def on_add(sel):
    sel.annotation.set_text(f"({sel.target[0]:.2f}, {sel.target[1]:.2f})")

plt.show()
```


Semi Supervised Fuzzy Clustering
Clusters
Cluster 1
Cluster 2
Cluster 3
Cluster 4
UMAP-2
UMAP-1
12.5
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5