Foundations of Artificial Intelligence
Joel Mattsson - Assignment 3
2024/2025

1. Problem Introduction

This report covers models using unsupervised learning to discover underlying patterns of data to partition them, based on inherent feature similarities, into clusters. The focus is directed toward using the MNIST database in an efficient manner, while not compromising too much effectiveness. The problem to be solved regards partitioning the MNIST database of digits from 0-9 into high quality clusters. To make this possible, three clustering approaches are implemented along with a dimensionality reduction technique and an evaluation formula for the qualities of the produced clusters. Below are the two primary topics addressed:

- Unsupervised learning with clustering: Exploratory discovery of underlying patterns in the dataset. Examines relationships and similarities between features and groups the data points accordingly. The ones that are similar to each other form dense regions denoted as clusters
- Dimensionality Reduction: By applying preprocessing methods to simplify the dataset before the model uses it, the quantity of computational resources needed by the model is reduced. For this reason, the time and memory required for the model to operate successfully decreases, making it more efficient. However, this begs the questions:
    - *At what cost? What is compromised? How significant is the negative impact on the output accuracy of the model?*

Both of these aspects are broad and crucial in machine learning. The larger the dataset (or the more dimensions), the more instances it can be used to train on, and thus the more accurate it becomes since the pattern becomes more clear. On the other hand, the drawback is the performance. Larger dataset also implies longer processing times and computational resources. In addition, not all data-columns or features contribute equally to the final output of the model. Some may be more valuable to examine, whereas others possess no influence over the model's predictions. In this way, there is a tradeoff between efficiency and effectiveness. Another aspect of the problem to be solved is predicated on this very tradeoff. In order to gain a bigger picture of the impact of the reduction of dimensionality, a factor or adjustment variable to regulate the levels of its application enables a comparison of

- *The elevated efficiency followed by a lower accuracy*
- *The lowered efficiency followed by a higher accuracy*

Thus, another part of the problem to solve is assessing the impact that dimensionalities of a dataset brings to the overall performance and runtime of a model. For this project, the MNIST database is fetched, with 70000 grayscale images in total of handwritten digits. It's important to note that it offers a wide variety of data instances with many dimensionalities and complexities. As a result, the adjustment variable will be effective in bringing insightful findings. The two other types of adjustment variables are denoted as *K* and *Kernel-Width*. Both of these are associated with the cluster methods' way of expanding or diminishing clusters, which make them valuable to investigate for further conclusions.

2. Theoretical Background

To fully grasp how the problem presented in the previous section will be solved, a few topics that are related to the implementations put forth in the next section “Approaches” will be discussed.

2.1 Machine Learning Models

In simple terms, these models are mathematical programs that learn from data to produce output when given an input instance that they aren’t explicitly programmed to encounter. This process is sequential. First of all, input data is needed for the model to learn to recognize patterns between the contained features. Secondly, after the training, the parameters need to be modified to yield better results. And lastly, the model’s performance is tested. In cases where unsatisfying predictions are made, you can either examine the way the model learns the patterns or adjust its complexity through the parameter values. Models are generally divided into three different types, based on how they process and interact with the input data where pattern recognition is established:

- Supervised Learning: Models that use supervised learning derive patterns from labeled datasets, where each instance is coupled with a value. The pattern for the model to internalize is thus explicitly apparent.
- Unsupervised Learning: No labels are present in the input dataset, and the model has to explore and discover the underlying feature similarities to then group them into clusters. These clusters are therefore representing the discovered patterns
- Reinforcement Learning: Through interacting with an environment, an agent progressively learns to make decisions in a process of trial and error. The actions are either rewarded or penalized, and the agent uses this knowledge to adjust its actions accordingly. In this way, the rewards are maximized and its decision-making optimized for the scenario or environment

2.1.2 Generalization Errors

In machine learning, generalization errors are inevitable. As the foundational idea of a model is to perform well on unseen data, or to generalize a pattern that properly describes the difference between all instances, this issue is present for all models. To account for cases that the model isn’t explicitly programmed to deal with is the main goal, and as a result, it must adapt a recognition ability that correctly identifies dependencies between features. The question then becomes:

- *In regards to the model’s recognition ability, how precisely should it fit to the training set? What is the optimal balance between bias and variance?*

If the model fits the training data too well, it tends to overfit, as it learns to capture noise that doesn’t belong to the actual desired pattern. As opposed to overfitting with a high variance,

models underfit when the bias is high. It's important to note that these types of generalization errors appear in both supervised- and unsupervised-learning contexts: with no regard to whether labels are included or excluded, to whether training and test sets are included or excluded, or to the model's assigned task: learning a pattern via ground truths or discovering new ones through exploration. Two common methodologies for improving the generalization are hyperparameter tuning and cross validation. In unsupervised clustering, for instance, hyperparameters govern the complexity of the model. In most cases, increasing a parameter entails an elevated sensitivity of the model, such that it picks up more variance and details. In contrast, decreasing a parameter's value would make a model more simplistic, causing it to make stronger assumptions with higher bias, exposing the risk of underfitting. On the other hand, cross validation is more suitable for label-based supervised learning methods, by virtue of its dataset division into train- and test-sets. As opposed to optimizing the model through parameters, it provides a reliable measure of the model's performance. It serves the purpose of evaluating the model, using different combinatorial folds to present a comprehensive estimate that more closely aligns with the actual truth of the dataset rather than relying on how well arbitrarily selected data conforms to the entire dataset.

2.2 Dimensionality Reduction

In essence, dimensionality reduction techniques are concerned with minimizing the number of dimensionalities in a given dataset. Since each dimension brings additional complexity and thus more computational power, this practice makes the processing of the data more efficient. However, the challenge lies in maintaining the level of information. Eliminating dimensions entails that the corresponding relationships also are excluded. This begs the questions:

- *To what degree does this practice negatively impact the output accuracy of the models that utilize dimensionality reduction?*
- *Or, does it even lead to a worse output accuracy? Is it possible to maintain effectiveness while increasing efficiency?*

Below, two techniques that are distinguished by how they transform data are addressed. Both approaches can be applied depending on the nature of the input data and the particular objective of the dimensionality reduction in the given context.

2.2.1 Feature Selection

As the name implies, this technique is predicated on the selection of features in the input dataset. In terms of the importance or influence a feature has for the prediction accuracy of the model, it systematically chooses a subset of fundamental feature-dimensions while neglecting the ones that are redundant.

There are three types of feature selections:

- Filter Methods: Rank all features by statistical metrics and only select the ones with the highest ranks
- Wrapper Methods: Assess feature combinations and interactions using predictive methods
- Embedded Methods: Covers cases where feature selection is integrated into the training session of a model

All three methods above serve the purpose of selecting appropriate features. Their use cases differ and are applicable to different scenarios, depending on the relationships of the dataset's features. For instance, filter methods are more efficient but neglect feature dependencies, and wrapper methods are computationally expensive but involve feature interactions.

2.2.2 Feature Extraction

As opposed to feature selection that avoids altering the features and solely reduces their count, feature extraction transforms the data into a simplified lower-dimensional representation. Rather than returning a subset of the original features, it composes new ones that contain the meaning of the original input data.

Types of feature extractions:

- Linear: Works best on more simplistic datasets with linear relationships between features
- Non-linear: Takes care of more complex data, where the relationships constitute non-linearities
- Hybrid: Combines both approaches to alleviate the limitations of implementing them separately. It integrates elements by sequentially applying them or combining their features during runtime

2.3 Clustering Model Types

With consideration to the stated problem in the section "Problem Introduction", an examination of unsupervised clustering types is necessary. The theoretical observations are categorized into four different types and aims to highlight a discernment based on how clusters are processed.

2.3.1 Partition-based

This clustering technique partitions the dataset into subsets representing clusters, striving to maximize the inter-clustering and minimize intra-clustering, such that all data points for a cluster are closely distributed to each other, while maintaining a large distance to other clusters. In this way, the process of discerning between inherent cluster qualities is facilitated:

- High external cluster dissimilarity: Ensures that no clusters are overlapping, which would cause confusion
- High internal cluster similarity: Highlights the significance of features and their contribution to the data point's final cluster outcome

Furthermore, partition-based clustering techniques require a predefined number of clusters to be specified before runtime, and they combine assignments of data points and iterative refinement with centroids to form clusters. A centroid is the average position of all data points within a specific cluster, and serves as the reference point for assigning data points to clusters. However, the shape of these clusters are limited to only spherical ones, and arbitrary shaped clusters are not supported, which may not hold for all datasets. While partition-based techniques have a simplistic and straightforward nature that's easy to interpret, it is dependent on the initialization of centroids. In the early stages of the algorithms, centroids are assigned an initial placement with limited information about their optimal location with respect to the dataset. This can lead to local optimum, where the best global solution cannot be reached.

2.3.2 Density-based

This technique focuses on grouping data points based on density. Regions of higher density indicate the potential for a cluster to be formed, whereas regions with low density are processed as noise or outliers. Two important hyperparameters for tuning the model for optimal generalization are:

- Epsilon: The maximum distance between two points for them to be considered as part of the same dense region
- minPts: The minimum number of points required to form a cluster or dense region

Using the concept of density of data points along with the hyperparameters, the following categorizations are made during runtime to form clusters:

- Noise Point: A data point in a separated region of low density resembling an outlier
- Core Point: A data point that, within its radius *Epsilon*, has at least a number of *minPts* data points. This point is considered to have a somewhat central position in the cluster (close to the core), as it's located in a dense region
- Border Point: A data point inside of a *Core Point's* dense region, but lacks enough neighboring points to satisfy the criteria of *minPts*. This implies that the data point has the outermost location in a cluster, delimiting the cluster's outer border

In comparison to partition-based techniques, there are a few advantages. The filtering of outliers is effective, as low density regions are systematically disregarded. In addition, the high density grouping allows for clusters of irregular shapes and distributions, extending beyond the limited scope of spherical ones with even distribution. Lastly, the flexible nature of density regions also removes the need for specifying a predefined number of clusters. With that said, density-based techniques may be more complex, but they provide more flexibility to the types of datasets it is applicable to. However, the performance is strongly linked to the

choice of *Epsilon* and *minPts* parameters. Therefore, density-based techniques are usually coupled with hyperparameter tuning.

2.3.3 Model-based

This type of clustering approach uses likelihood estimates to assign data points to clusters. By applying a probabilistic framework that assumes that a cluster is represented by a probability distribution, a data point can belong to multiple clusters with varying degrees of membership, based on the confidence of the prediction. In essence, model-based techniques first calculate a data point's associated probability of belonging to a cluster, using parameters such as covariances and means. Then, it proceeds with adjusting the parameters to increase the likelihood. These two steps are repeated until the obtained probability crosses a threshold, or until a convergence is reached. A significant benefit that this type of clustering approach yields is the flexibility of cluster properties that can be identified. As mentioned, density-based clustering is flexible in comparison to partition-based due to its ability to manage clusters of arbitrary shapes and disregard noisy data. However, it struggles when the clusters have varying degrees of densities. Model-based clustering, on the other hand, is suitable in this scenario, leveraging underlying probabilistic distributions to alleviate constraints and further improve flexibility. Even though benefits are derived from the probabilistic framework, it also comes with drawbacks. The probability calculations are computationally expensive, making it unfavorable for large datasets. It also requires a predefined specification of the number of clusters, just like partition-based clustering methods.

2.3.4 Graph-based

As the name implies, this technique maps data to a graph, turning data points into nodes or vertices, and relationships into edges. The edges connect data points and are weighted to indicate similarity or distance between them. In this graph, a cluster is a subgraph with more densely connected nodes, such that the contained edges demonstrate higher similarity. As opposed to density-based clustering, graph-based may struggle to successfully discern between noise or outliers and sparse clusters. Since a graph-based cluster is a subgraph of close edges, a meaningful cluster of a sparse subgraph may wrongfully be detected as noise. However, this boils down to the configured hyperparameters and threshold values, which can be optimized in accordance with the given dataset's inherent complexities and feature relationships. Hence, the performance is sensitive to parameters. On the other hand, as the method relies on edge-weights, it makes no assumptions about cluster shapes and instead focuses purely on the underlying information of the feature relationships. Although graphs remove cluster forming restrictions, their construction and management bring computational challenges. The process of mapping each data point or node to distance-based edges requires each feature relationship in the dataset to be quantified. As a result, high-dimensional and large datasets significantly diminish the efficiency.

3. Approaches

As discussed in the previous sections, due to the nature of the problem to be solved that calls for unsupervised learning along with clustering approaches, all implementations presented below are concerned with exploring underlying relationships and data partitioning.

3.1 Principal Component Analysis

Principal Component Analysis (PCA) is a dimensionality reduction technique, and is considered unsupervised learning because of its way of disregarding labels and focusing on exploring feature relationships. This technique is categorized as a *Feature Extraction* dimensionality reduction, as mentioned in the previous section "Theoretical Background". One important question that naturally arise around this approach is:

- *If PCA isn't directly about improving the model's effectiveness, as its sole purpose is concerned with dataset simplification, then, why even go through the hassle of implementing this approach?*

In this project, PCA is used as a preprocessing step before applying the clustering methods. In other words, PCA is the first step in the pipeline that takes the MNIST dataset as input and produces a simplified version of it, before passing it to the next filter or method. Moreover, it brings multiple worthwhile benefits:

- Interpretability: With dismissed redundancies, the model can interpret the dataset at ease
- Model Efficiency: Highlights important feature relationships with highest contribution to the clustering. In this way, the model's processing of data requires less computational resources
- Generalization: The reduced number of features helps eliminate less informative dimensions and noise. This simpler representation allows the model to focus on more significant patterns rather than learning irrelevant noise and overfitting

All of these listed benefits are a product of the algorithmic principal components. In essence, PCA uses a covariance matrix to compute eigenvalues and eigenvectors. These vectors indicate the directions of principal components, and the eigenvalues determine their magnitude. The principal components are then filtered based on the highest eigenvalues, and the data is then transformed into a lower-dimensional space using a matrix of the associated eigenvectors. With this simplified representation, the complexity can be adjusted to avoid overfitting. This ties into what was stated in the "Theoretical Background" section with overfitting and underfitting: to balance the complexity of the model and the data to minimize the generalization error. As for PCA, we want to minimize the gap between the model's sensitivity or complexity, and the contained complexity of the dataset and its dimensionalities. In other words, we want a higher compatibility of the input dataset with the model, so that they fit or match more closely, yielding a more favorable performance. With that said, it's not only the model that we can adjust, but the input dataset as well. Only

focusing on the efficiency of the model is therefore a limited and one-sided approach, disregarding possibilities of refinement that a versatile approach would bring.

As the model forms a perception of patterns to recognize from the dataset, the dataset becomes crucial. If it is erroneous, the model also adapts an erroneous pattern, even if the model was mathematically implemented correctly. The question then becomes:

- *If the input can be simplified, how can it systematically be deconstructed or decomposed, while retaining its core patterns that are crucial to the model's performance?*

As addressed earlier, eigenvalues and eigenvectors and principal components are used to identify the most influential feature combinations: to condense or simplify the dataset without losing critical insight. In this way, the features are selectively simplified with respect to the preservation of the important relationships that make up the clusters. Furthermore, this very quality of PCA is also useful for noise reduction, since the irrelevant or insignificant feature combinations are detected. As mentioned in "Theoretical Background", partition-based and graph-based clustering techniques struggle with detecting noise properly, which is detrimental to their performance. Consequently, applying PCA in these cases would filter out noise or flaws that would otherwise cause inconsistencies in the clustering. Yet, PCA isn't ideal for all cases. In situations with low-dimensional data, there's no need for additional transformation to a further lower dimensionality. It not only becomes redundant, but information is lost. Loss of information is a general assumption for PCA, and this reduction of details, for simple datasets, is exposed to the risk of removing important details. The question then becomes:

- *When does the cost of the guaranteed loss of information exceed the benefits of a resulting simplified dataset? To what degree of complexity or dimensionality must a dataset obtain, such that applying PCA becomes favorable, despite the fact of losing information?*

The solution would be the integration of a parameter that adjusts the applied dimensionality reduction level to the dataset, to minimize the gap between the model's bias and variance and the dataset's inherent complexity:

- <u>If the model's poor performance is caused by overfitting due to the dataset being too simple in relative to the model's complexity:</u> Reduce PCA level, providing a transformed dataset closer to the original dataset's complexity
- <u>If the model's poor performance caused by underfitting due to the dataset being too complex in relative to the model's complexity:</u> Increase PCA level, providing a more thoroughly preprocessed and transformed dataset, better matching the variance and bias of the model

In this project, this parameter controlling the PCA level was implemented. Due to the complexity of the MNIST database, the concept of dimensionality reduction is significantly beneficial. However, since three different clustering method approaches also were implemented, the need for modifying the applied intensity or effect of PCA emerged. Since

all model's have their own complexities and have different levels of sensitivity, they respond to varying levels of bias and variance. Consequently, it's important to adjust the PCA level to optimize generalization for unseen data in this project, despite the fact of using the same dataset for all models.

3.2 Gaussian Mixture Model

This type of approach belongs to Model-based clustering, as mentioned in the previous section "Theoretical Background". It assumes an underlying statistical model to identify clusters and that data is generated from normal distributions. Gaussian Mixture Models are known for their probabilistic qualities made possible as a result of the mean (center of a cluster) and covariance matrix (shape, size and orientation of a cluster). By iteratively utilizing expectation and maximization steps, it continuously optimizes the probability predictions in the following way:

- Expectation Step: Calculate probability that a data point belongs to a cluster
- Maximization Step: Update parameters of distributions in accordance with the calculated probabilities in the expectation step

Keep in mind that, for the expectation step, a data point can belong to multiple clusters due to the expression of probabilities. However, the one with the highest confidence or probability estimate is the one most likely to be the data point's one and only cluster. In this way, the assignments are soft and nuanced, as opposed to directly attaching a data point to one cluster immediately.This is why the reiteration of these steps is critical. The worst case is where the expectation step computes equal probabilities for a data point to belong to multiple clusters, since this scenario doesn't favor its tendency or compatibility to any cluster. The goal is to iterate the probability computations and the updating of parameters until a point of convergence or satisfaction, such that the computed probabilities have high confidence in their clustering predictions. In other words, each data point's probabilities attached to each cluster should be relatively low, except for its actual cluster that it belongs to. This makes it apparent to discern or make an accurate decision. Using this unique probabilistic approach yields several advantages:

- Cluster shape flexibility: Unrestricted shapes are detected, unlike partition-based clustering methods
- Overlapping clusters: Successfully handles overlapping clusters, combining soft assignments with probabilities
- Handle noise: Noise and outliers are assigned lower probabilities and detected with ease

The incorporation of a probabilistic framework opens up many possibilities. Besides the flexibility of cluster shapes, the ability to register clusters even though they are overlapping is made possible. As each data point has associated probabilities to each cluster, the model is able to recognize patterns and dependencies to other clusters. In more detail, each cluster is a distribution with a mean and covariance matrix, allowing an independence of inter-cluster

quality, dismissing the confusion of a data point's clustering if two clusters have a high similarity. Nonetheless, gaussian mixture models have a few limitations:

- Computationally expensive: Compute probability for all clusters for each and every data point and then reiterate the assignments while updating parameters
- Assumes a gaussian distribution: The model may struggle to fit the data accurately if it deviates from the gaussian distribution, as it is assumed to represent data in each cluster
- Predefined number of clusters: If the number is not chosen correctly, the resulting clustering quality may suffer. This is similar to the drawback of partition-based clustering techniques, where the number *k* must be specified in advance, significantly impacting the results

Although all of the points above are negative, there are ways to counteract or mitigate them. As for the point about expensive computations, it was mitigated by implementing PCA and simplifying the data from MNIST database, before applying Gaussian Mixture Model clustering on the data. This approach also assumes a gaussian distribution, which indicates a lower performance for datasets not complying to this standard. However, in general, this distribution is a natural occurrence in processes as it is based on the central limit theorem, stating that adding random variables together yields a sum that tends toward a normal distribution. And lastly, concerning the predefined number of clusters, we can find optimized values using methods such as the Elbow Method and Bayesian Information Criterion.

3.3 Mean Shift

This type of clustering is categorized as density-based, as discussed in the "Theoretical Background" section. The unique characteristic of mean shift is its ability to identify clusters or dense regions in the absence of explicit supervised labels, while estimating density of data points. In essence, it uses the dimensionalities or features in a space and shifts data points to higher density regions. These dense regions are marked as potential clusters during execution, and are at the final step, before the algorithm terminates, labeled as a cluster or noise, with respect to the shifted mean. This clustering method is iterative in nature and follows the outlined process below:

1. Start with a random data point
2. Initialize a kernel around the selected point that contains other data points within the specified radius known as kernel width or bandwidth
3. Calculate the mean of all data points within the radius or kernel's range
4. Shift the kernel's radius area to the mean of the points. In other words, move the center of the kernel's range or bandwidth, within the discovered dense region, to the new center of this region (keep in mind, this is marked as a potential cluster during runtime)
5. Reiterate this process as the distance of the shift approaches zero, indicating that the absolute center of the dense region has been found, where no other potential data points could successfully be included in this region

Eventually, the algorithm reaches a state of minimal shift movement, whether that be no movement at all, or exceeding a certain threshold, causing it to exit the iteration and finalize the clusters. This termination may be triggered by different exit conditions, and this has a profound impact on the performance:

- After a number of iterations: Worst case, since the ultimate goal is to minimize the shift movements
- After the shift movement is consistently below a threshold: Average case
- After the shift movement consistently is freezed, indicating no movement at all: Best case, most distinguishing cluster results

Reaching a state of minimal shift movement essentially indicates that a cluster has been identified. The smaller the movement is, the more defined the clusters. Conversely, if no progression is made, such that the shift movement's high rate persists throughout the execution, then the only viable exit condition is an iteration limit. This implies that the cluster, based on the data distribution and bandwidth, wasn't validly detected. Although there's a strong dependency between the given data's distribution and the triggered exit condition, the implications that the bandwidth radius parameter brings are simultaneously positive:

- Does not require the number of clusters: Utilizes the radius parameter instead of requiring the number of clusters upfront
- Clusters of arbitrary shapes: Rather than relying on spherical shapes, density regions around the kernel-width radius is used to signify clusters
- Less sensitive to outliers: The density estimation identifies sparse areas as noise

On the other hand, Mean Shift also has some disadvantages:

- Computationally expensive: Its iterative nature of density estimations is a significant burden in the computational domain
- Bandwidth: Although this parameter brings many positive benefits, it comes with one negative aspect. Controlling the division or merging of clusters is difficult when the radius is the determining parameter instead of the number of clusters. As a result, you may get too many small clusters, or just a few large ones that are merged from distinct clusters. This all depends on the distribution of data and the dense regions. In other words, the bandwidth parameter yields an unpredictable or inconsistent performance depending on the contained distribution or relationships of the given dataset.

As mentioned in the point above, the negative aspect of bandwidth is the unpredictability of performance together with a dataset's density distribution. However, this parameter can be optimized to produce higher quality clusters, by examining how it functionally integrates with the mean shift algorithm for optimal results. The *bandwidth* parameter resembles the *k* value determining the number of clusters for many other clustering methods. Although the bandwidth doesn't directly allow us to know in advance how many clusters will be created, it affects how many data partitions are classified as clusters. Instead of representing an upfront value for the number of clusters in the output, the bandwidth parameter governs the radius

size of the kernel's width. In essence, this impacts the entire process of detecting dense regions since either more or less data points are included or excluded from their regions. Ultimately, the calculation of means for these regions are affected, and in this way the shift or movement of the kernel's radius (the cluster's position) is impacted. Consequently, the sizes and positioning of the clusters are changed as the bandwidth changes, potentially also changing the number of clusters in the final output.

3.4 Normalized Cut

Normalized cut belongs to Graph-Based clustering, as mentioned in the "Theoretical Background" section. In an exploratory fashion, it discovered structures or underlying patterns through unsupervised learning. This is done by dividing data into clusters based on graph properties, transforming data points into vertices and their corresponding relationships into edges. It's important to note that each edge has an associated weight that measures how similar two data points, or nodes, are to one another. Using this similarity concept, nodes are grouped in accordance with how densely connected they are, ultimately forming clusters. Normalized cut performs a sequence of operations to make this implementation possible:

1. Graph Representation: Nodes represents the data points and an edge connects a pair of nodes with an associated weight, constituting the similarity of the two nodes
2. Graph Partitioning (define clusters): Divide the graph into two separate subsets, where all nodes sharing the same subset are very similar to each other, whereas nodes in different subsets have a low similarity
3. Cut Metric: A cut between the two subsets uses the total weight of all edges for all connected nodes between the two subsets. The problem is that this one-sided approach isolated nodes from the rest of the graph, leading to imbalanced clusters
4. Normalized Cut (balance clusters): This kind of cut uses the aforementioned cut metric together with overall associations of all subsets of the graph. In essence, it makes sure that clusters are formed with high intra-clustering quality with balanced sizes by penalizing unbalanced partitions
5. Optimization: As a result of Normalized Cut's inclusion of subset association, it becomes combinatorial in nature. This is computationally expensive, and is alleviated by using eigenvectors for a low-dimensional embedding of the data, preserving the graph structure

As it appears, a cluster is a subgraph with more densely connected nodes, such that the contained edges demonstrate higher similarity. By extension, this approach is powerful when relationships between data points are more informative than the data itself, leveraging on data that naturally forms a network. Moreover, the key characteristics of this clustering technique lies in its cutting features. First of all, it uses a cut metric to evaluate the total weight of edges that are cut at the divergence of two subsets. Secondly, the cut is normalized by applying associations of subsets:

- Cut Metric: Quantifies the cost of separation between two subsets
    - ***A:*** *The first subset*

- ***B:*** *The second subset*
- ***w(i, j):*** *The weight of the edge between vertices i and j*

$$\text{Cut}(A, B) = \sum_{i \in A, j \in B} w(i, j)$$

- Normalized Cut: Achieve partitions with minimal separation cost (cost metric), yet obtain strong connectivity
  - **Cut(A, B):** The numerical value of the cut metric of two subsets
  - **V:** The set of vertices
  - **Assoc(X, V):** The total connection of data points in subset **X** to the entire graph or dataset

$$\text{Ncut}(A, B) = \frac{\text{cut}(A, B)}{\text{assoc}(A, V)} + \frac{\text{cut}(A, B)}{\text{assoc}(B, V)}$$

In greater detail, the normalized cut makes the following achievements possible:

- Balances clusters: The cut normalization with subset associations prevents isolated and imbalanced clusters
- Captures entire graph structure: The normalization and associations ensures the entire structure and overall data distribution is captured
- Output cluster qualities: This approach also emphasizes the concepts of intra-clustering and inter-clustering to produce higher quality clusters. High similarity of data points within the same clusters and low similarity between distinct clusters is prioritized. This makes it easier to distinguish between each data point's final cluster

Furthermore, Normalized Cut brings other aspects as well. As the technique relies on edge-weights, it makes no assumptions about cluster shapes and instead focuses purely on the underlying information of the feature relationships. Although graphs remove cluster forming restrictions, their construction and management bring computational challenges. The process of mapping each data point or node to distance-based edges requires each feature relationship in the dataset to be quantified. As a result, high-dimensional and large datasets significantly diminish the efficiency. In addition, as listed in the algorithmic sequential steps above, the optimization is an extra step implemented primarily for the sake of maintaining an acceptable level of efficiency, since the combinatorial association of normalized cut is computationally expensive. Yet, this extra step still brings further computational challenges and requires more resources.

3.5 Rand Index

In the broad topic of machine learning, it's fundamental to be able to measure the actual performance of a model against its intended behavior. Since this topic is vast, there are multiple sub branches that process and manage data very differently. For instance, in

classification, the model is trained to recognize a pattern, and the evaluation is therefore based on the correctness of the predictions. Metrics like accuracy, precision, recall and F1 score are relevant for this approach. However, in the scope of this project, with respect to the problem addressed in "Problem Introduction", cluster evaluation is more appropriate. As opposed to classification that relies on predictions and labels, clustering evaluation aims to measure the clustering quality. It measures the compactness or similarity of the data points inside of a cluster as well as the dissimilarity between that cluster and other clusters. Rand Index is a numerical representation of how well data points are clustered, and it ranges from 0 to 1. What makes this method unique is its usage of a ground truth coupled with the actual clustering of a data point. In this way, a pairwise evaluation is used to check if they are effectively grouped in alignment with their intended cluster. Below, the implemented formula is shown:

$$R = \frac{2(a+b)}{n(n-1)}$$

Where..

- ***n:** Number of datapoints in dataset*
- ***a:** Number of pairs representing same digit AND are clustered together (true positive pairs)*
- ***b:** Number of pairs representing different digits AND clustered in different clusters (true negative pairs)*

By virtue of both *a* and *b* indicating positive qualities, and the fact that they are in the numerator in the formula, the following conclusion can be made:

- *The higher the values for **a** and **b**, the higher the rand index score*

This makes sense as *a* essentially ensures a high intra-clustering quality: that all data points are in their intended clusters, whereas *b* is responsible for the separation from incorrect clusters.

Advantages of Rand Index:

- Intuitive and easy to interpret: Pairwise comparison between predicted clustering and ground truth labels
- Balanced evaluation: Considers both pairs that are grouped together and pairs that are separated into different clusters

Disadvantages of Rand Index:

- Requires ground truth labels: Ground truth labels may not always be available for unsupervised datasets
- Limitations in addressing biases: As the Rand Index tends toward an average calculation, it doesn't address cases where clusters have significantly different sizes. For instance, a large cluster contributes more to the final Rand Index score, as it has

more pairwise comparisons. In this way, smaller clusters are overshadowed and the Rand Index disproportionately represents cluster qualities based on their size

4. Results

This section solely intends to present the results briefly. The results below were obtained through JSON files generated during runtime for each of the approaches described in the previous section. The rationale and conclusions derived from these insights will be presented in the next section “Conclusions”. Below, the two subsections are presented. The “Runtime Data” pertains to the JSON content generated during execution of the implemented approaches, and “Visualizations” is based on what the JSON data indicate, providing visual graphs to better illustrate insights. Note that the sections are divided into the three clustering approaches, and no section for the preprocessing step of PCA is included. This is due to its involvement in all three approaches. The data for its performance is associated with each execution of all implemented clustering methods, and will therefore be presented together with each technique.

4.1 Runtime Data

In this section, only JSON data with brief explanations of their implications are covered. It’s important to be familiar with the produced JSON data structure to understand how information during runtime is extracted. The produced structure is consistent across all three clustering methods. The only attributes that differ are the ones in *mean_shift.json*, where all of the *K*-values are replaced by values of the *Kernel-Width*. With that said, only the naming conventions of these two attributes are different. Below is a picture of the simplified structure used in this analysis.

Brief explanation of the JSON structure

- Combinatorial Configurations: These values are assigned at the start of the execution of the program based on what settings the developer selected
    - PCA Level **(i)**: The current PCA level to apply in the iteration
    - **X** Values **(j)**: *X* represents either *K* or *Kernel-Width*, depending on which clustering method is executed. This attribute is the current value obtained in the iteration for each *X* value
- Run Time Data: Container for all data generated during execution
    - Results: Array of all objects of applied PCA Levels, along with their inherent combinations of *X*
        - PCA Level = (**i**): Applied PCA Level
            - **X**-combinations: Array of objects containing performances of current *X* value
                - **X = (j)**: The current applied value (K or Kernel-Width)
                - Rand Index Performance: The associated achieved score
                - Time Elapsed: The time elapsed in seconds

For additional information about the JSON structures, navigate to *schemas.yml*, which contains sub-schemas divided into K-value and Kernel-Width types. In essence, the corresponding clustering method attaches sub-schemas from this file into the produced JSON objects which are then filled with the performance values during runtime. Moreover, in the sections below, the pictures aren't able to capture the entire JSON files, as their lengths range from 100-300 lines. Instead, snippets show a fraction of the contained data, which are then followed by tables that are based on data from the entire files.

4.1.1 Gaussian Mixture Model

The data demonstrated in this section are derived from */output/data/gmm.json*. The configured settings can be found in the "Combinatorial Combinations" attribute which indicate that the PCA levels are [10, 50, 100, 200] and the K-values range from 5 to 15. Also, under "Run Time Data", the two uppermost attributes entail that the total execution time for 10000 instances (a subset of MNIST database) was 1043 seconds.

In the tables below, all of the generated data in *gmm.json* is presented in a structured way.

PCA Level = 10 (lines 29-87 in the submitted json file):

Parameter	Value
Time Elapsed	251 seconds
Rand Index	Min: 75% (K = 5), Max: 87.7% (K = 11)
Notes	The only K value that yields a Rand Index below 80% when the PCA level is 10 is the lowest one, at K = 5. Increasing K seems to generally increase the Rand Index score in this case, as for K >= 9 the Rand Index is greater than 85%

PCA Level = 50 (lines 90-149 in the json file):

Parameter	Value
Time Elapsed	272 seconds

Rand Index	Min: 69% (K = 5), Max: 84% (K = 15)
Notes	Once again, we see a general trend where the Rand Index score increases in parallel with the K value

PCA Level = 100 (lines 151-210 in the json file):

Parameter	Value
Time Elapsed	259 seconds
Rand Index	Min: 67.8% (K = 5), Max: 83.9% (K = 13)
Notes	For K = 15, the score was close to the highest, at 83%. The same pattern of higher K values is recognized, where the lowest value for K yields the minimal score

PCA Level = 200 (lines 212-270 in the json file):

Parameter	Value
Time Elapsed	260 seconds
Rand Index	Min: 66% (K = 5), Max: 81.9% (K = 15)
Notes	The pattern is replicated once again, where the minimal and maximal K values produces Rand Index scores in proportion to the size of K

General Notes:

- For each PCA Level, decreasing K decreases the corresponding Rand Index. This trend appears to be strict, as the lowest Rand Index for each PCA Level is obtained when K is assigned a low value, whereas higher K values consistently produce more satisfying scores

- Increasing the PCA Level degrades the Rand Index scores. When it was set to 10, the highest Rand Index for both the minimal and maximal aspect were achieved. On top of that, it was executed in the quickest fashion, only requiring 251 seconds for 10000 instances in MNIST database

4.1.2 Mean Shift

The data below is derived from */output/data/mean_shift.json*, and it uses Kernel-Width along with different PCA Levels to produce the presented data.

PCA Level = 10 (lines 22-46 in the json file):

Parameter	Value
Time Elapsed	118 seconds
Rand Index	Min: 89.9% (Kernel Width = 0.5), Max: 90% (Kernel Width = 2.0)
Notes	The variance between each output is significantly low, where the min and max scores are separated by a 0.1% difference. Increasing the Kernel Width slightly increases the rand index score in parallel

PCA Level = 50 (lines 48-71 in the json file):

Parameter	Value

Time Elapsed	201 seconds
Rand Index	Min: 89.9% (Kernel Width = 0.5), Max: 89.9% (Kernel Width = 2.0)
Notes	All rand index scores are tightly ranged within 89.9%, where only a fraction of a percentage separates each kernel width's performance

For the simplicity, interpretability and readability of this report, the two other PCA Level analyses are excluded, since they indicate the same pattern as the two tables above. The general takeaways from Mean Shift's presented output data are:

- The Rand Index score for each PCA Level combination with Kernel Width is roughly the same 89-90%, indicating a consistency
- However, when increasing the PCA Level, rather than impacting the clustering quality score, it negatively influences the computational resources. When observing the time elapsed attribute for each PCA table above, for each PCA increment, a significant number of additional seconds is needed for the model to complete the clustering: starting at PCA Level = 10 with 118 seconds, and ending at PCA Level = 200 with 274 seconds. The interesting part is that the Rand Index scores are maintained, all consistently falling within the 89th percentile

4.1.3 Normalized Cut

The data below is derived from */output/data/normalized_cut.json*, and it uses K-values along with different PCA Levels to produce the presented data.

PCA Level = 10 (lines 28-87 in the json file):

Parameter	Value
Time Elapsed	296 seconds
Rand Index	Min: 73.8% (K = 5), Max: 84% (K = 15)
Notes	As K increases, the time elapsed and rand index score tends to increase

PCA Level = 50 (lines 90-148 in the json file):

Parameter	Value
Time Elapsed	296 seconds
Rand Index	Min: 73.9% (K = 5), Max: 84.4% (K = 15)

Notes	Once again, greater K implies greater rand index score

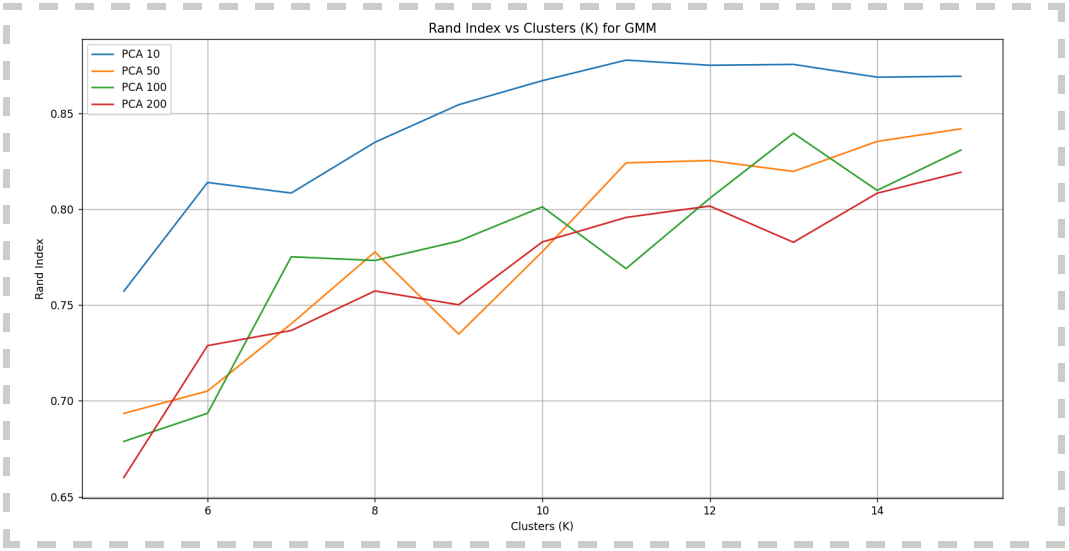
For simplicity of this report, the last two PCA levels (100, 200) are excluded. They embody the same pattern, where greater K values yields better rand index score, followed by slightly longer execution durations. As noted, the results for each table are consistent with each other, always ranging from 73-84%, with K=5 being the minimal value, and K = 15 being the highest. However, as PCA = (100, 200), the time elapsed decreases slightly. At level 50, it takes 298 seconds, whereas level 100 and 200 takes 297 and 293 seconds respectively. Their rand index scores are also slightly decreased, but this only regards fractions of percentages.

4.2 Visualizations

In this section, different charts are visualized to more clearly highlight patterns of the different implemented approaches. All of these charts use the Rand Index score as the y-axis to evaluate how well a clustering method performed during execution.

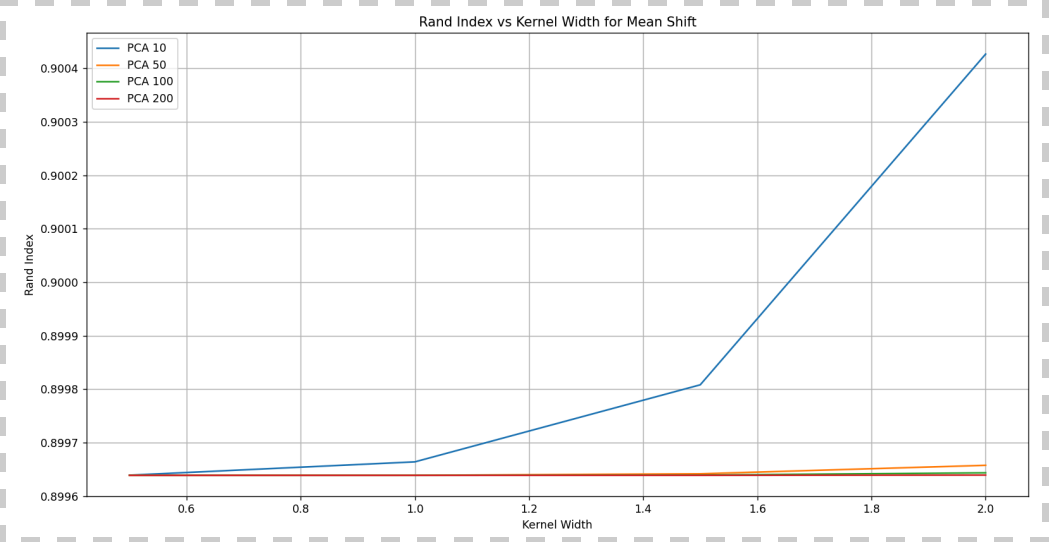
4.2.1 Gaussian Mixture Models

The chart below demonstrates the impact on the clustering quality that varying PCA level brings. Clearly, the lowest one, where the level is 10, is the best performing one. We also see that as the X-axis increases, so does the rand index score in general. This aligns with the insight that the JSON file pointed out: that larger K values yield better performance.



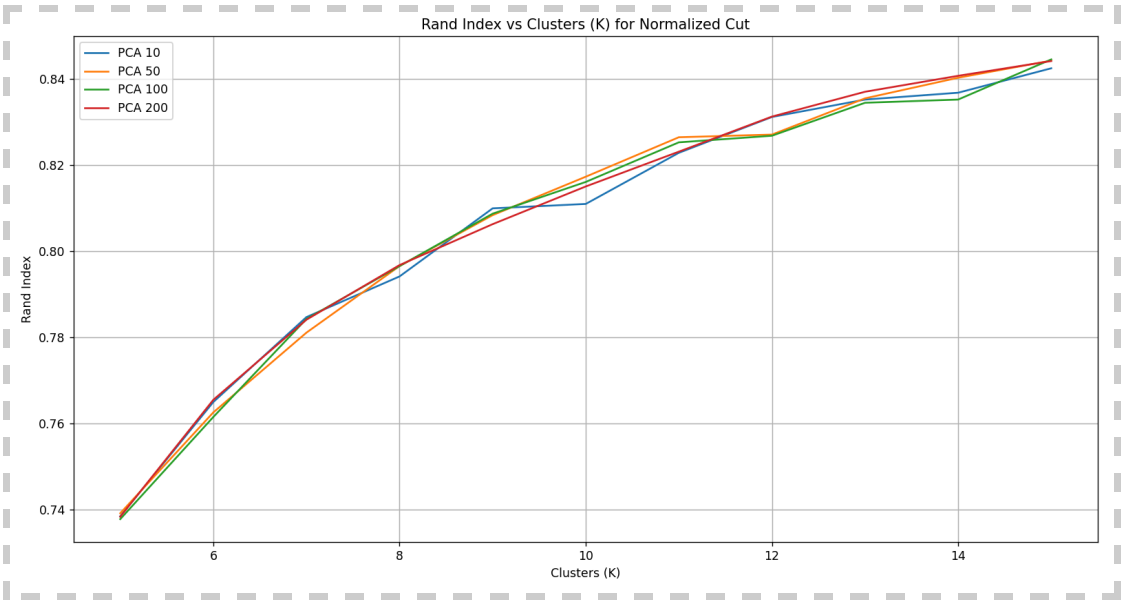
4.2.2 Mean Shift

The chart below illustrates that higher values for the Kernel Width impacts the performance positively. However, in this case of using the MNIST database, the optimal PCA level appears to be 10.



4.2.3 Normalized Cut

The chart below heavily implies a direct correlation between the Rand Index score and the K value. Since each line is tightly coupled, it signifies that this type of clustering method has a stronger indifference to the PCA Levels, as the performance remains unchanged.



5. Conclusions

After observing the results of the three different clustering methods coupled with dimensionality reduction, there appears to be many common threads to conclude. The conclusions in the following sections are related to the task of this project as well as the data obtained in the "Results" section: to partition handwritten digits in an efficient manner with optimal Rand Index score.

5.1 PCA Level significance to clustering performance

As the result section displays, PCA level 10 significantly outperforms the other levels for the clustering methods, particularly in the GMM and Mean Shift approaches. Since PCA is concerned with a dataset's complexity to reduce the dimensions, this brings insights into the data fetched from MNIST database as well as the associated clustering approaches. In this project, where 10 distinct numbers (0-9) are to be clustered, only a limited effort of noise reduction is needed. Additionally, since the digit images are highly structured, the dataset's intrinsic dimensionality is relatively low. Thus, lower PCA levels can still capture the variance in the dataset. In turn, higher PCA levels end up oversimplifying the dataset, struggling with preserving meaningful patterns. On the other hand, PCA with too few components may miss out on critical information, causing clusters failing to represent the true data structure. When examining the yielded results in the "Results" section, it appears that, the higher the PCA level, the worse the performance. This supports the conclusion that the 10-digit MNIST database has a structured and low-dimensional format that is interpreted by ease. It also entails that the clustering models' complexities are generally fitting better to more complex datasets, as their performances fluctuate in parallel with the simplicity of the transformed dataset. However, there's a clear disruption of this trend in Normalized Cut, where the PCA level has a much lower impact on the performance. This is because of the unique characteristics of the spectral methods that come with this approach. Its clustering performance depends more on the quality of the subgraphs rather than the dimensionality of the input or nodes, making it more independent to PCA. Ultimately, for all of the three implemented clustering methods in general, PCA level 10 appears to be the optimal one out of [10, 50, 100, 200], as 10 principal components better matches the clustering complexity that comes with 10 distinct digits from MNIST database.

5.2 K-value's impact on clustering performance

In the previous "Results" section, there's a strict trend that appears in every single visualization, which is also clearly identified in the JSON files: as the x-value grows, so does the Rand Index score. For this section that pertains to the K-value, the GMM and Normalized Cut approaches are looked further into. To begin with, the K-value represents the partitions or groupings in which the algorithm aims to return as distinct clusters. The essence is that, as the number of clusters is modified, the challenge that the algorithm faces is how it can optimally distinguish between each group. This is important as each grouping must have similarities in feature patterns. Thus, lower values for K indicate that a more shallow analysis of the data points' features is required, whereas higher values leave more room for differences and variances. The question then becomes:

- *What is the optimal value for K, given that the MNIST database with 10 distinct digits is fetched in this project?*

Well, it completely depends on the applied clustering method and how it operates with respect to the defined number of output partitions. Below are the two approaches that use *K* in their algorithmic calculations:

- Gaussian Mixture Model: The obtained results indicate that higher *K* allows for a more detailed separation of clusters, such that the model can better distinguish between the handwritten digits
- Normalized Cut: This method embodies the same pattern, where greater K yields better Rand Index score, as clusters are divided into more distinct partitions accounting for the variety of digit clusterings

In conclusion, lower values for K result in underfitting, as it makes stronger assumptions when the data is partitioned into less groups. This is due to the fact that fewer clusters implies that the similarity threshold is lower for all data points, and therefore the bias is elevated. Conversely, increasing K means that more clusters are to be formed, which forces the model to find more detailed patterns distinguishing each cluster. As a result, more variance is required to capture these patterns, often leading to overfitting. As for this particular project, where the MNIST database is used, coupled with GMM and Normalized Cut, smaller values for K would remove the possibility for 10 distinct digits or clusters. This explains the poor performance for lower K value that is observed in the "Results" section, where two or multiple digits represent the same cluster, ultimately producing low quality clusters, as reflected in the Rand Index scores.

5.3 Kernel-Width's impact on clustering performance

As for the Mean Shift approach, the trend of increased Rand Index score in parallel with kernel width is apparent. In the "Results" section, the values [0.5, 1.0, 1.5, 2.0] were used, and 2.0 delivered the highest satisfaction. To reiterate on the meaning of kernel width, it essentially represents the radar to which the mean calculation is performed, which influences how dense regions of potential clusters are discovered. The size of *Kernel-Width* has the following implications, considering its relation to the radar:

- The radar captures more or less data points in dense regions, based on its defined area. In this way, the forming of clusters is impacted
- The included or excluded data points in the radar defines the mean. In this way, the shift or movement of the center (the cluster's position) is influenced
- The radar registers more or less data in particular regions, depending on its size. In this way, the number of clusters is changed accordingly

Similar to the function that *K* serves, *Kernel-Width* controls the number of output clusters, but it does so in an indirect specification through radar size adjustment, as opposed to requiring a direct number upfront. Also, both variables influence the degree to which variance and bias are involved. The configurations that yield bigger but less clusters make stronger assumptions, whereas divided, distinct and many partitions imply more complexity to

distinguish between them. However, since their core meanings are distinct, so are their relations to bias and variance, although their fundamentals are aligned:

- Merge distinct clusters:
    - *Kernel-Width* increases
    - *K* decreases

- Divide big clusters:
    - *Kernel-Width* decreases
    - *K* increases

Conclusively, with consideration to the MNIST database and the 10 digits of partitioning, it is sensible to account for a variety of clusters yet still not limit their differences. Their distinct features must be correctly separated while not creating too many clusters, allowing data points to be assigned to inherently invalid partitions This problematic case occurred can be observed in the previous “Results” section for the lower values of *Kernel-Width*. As a result of too low *Kernel-Width*, too many small clusters were created and the model started overfitting. This explains why increasing this variable yielded higher Rand Index scores. On the flip side, increasing the *Kernel-Width* too much would induce underfitting. Ultimately, this would cause an oversimplification of the clusters, such that the desired clusters would be incorrectly merged into bigger partitions that represent a mixture of the actual outcomes.