WARSAW UNIVERSITY OF TECHNOLOGY

DISCIPLINE OF SCIENCE INFORMATION AND COMMUNICATION TECHNOLOGY
FIELD OF SCIENCE ENGINEERING AND TECHNOLOGY

Ph.D. Thesis

Mikołaj Markiewicz, M.Sc.

Evaluation of data partitioning strategies for distributed clustering and classification algorithms

Supervisor
Piotr Gawrysiak, Ph.D., D.Sc.

Additional supervisor
Jakub Koperwas, Ph.D.

WARSAW 2023

Abstract

The sizes of the various datasets collected worldwide are growing rapidly. These data are stored in different independent locations, and increasing numbers of new algorithms have been implemented to work with these distributed data, especially for classification and clustering tasks. However, there is no standardised way of validating such algorithms, and they are typically tested on independent and identically distributed (IID) data, which are uniformly distributed. The real data distribution across independent nodes is typically unknown, which affects the processing results. This work is devoted to improving the assessment of distributed algorithms by applying new non-IID partitioning methods to mitigate the impact of undefined data distributions. Moreover, a standard set of new partitioning strategies to simulate non-IID data distributions for the evaluation of such algorithms is proposed, and hidden shortcomings in algorithm processing results are revealed.

Additionally, the work introduces a new, easy-to-use and extendable platform with pluggable components to address the lack of comprehensive benchmarking tools in an actual distributed environment for distributed data mining (DDM) methods. The proposed platform enables a proper analysis of the results in terms of the multiple dimensions that impact the algorithm, such as the final quality, transfer load and detailed measurements of processing times for the different stages of processing.

Furthermore, this paper reveals the negative impact of various data distributions on the quality results of processing with distributed algorithms by evaluating a set of distributed classification and clustering methods. The author also introduces an adapted clustering hybrid algorithm that can achieve high quality and overcome the problem of specific data partitioning without the need for assumptions about a uniform data distribution.

Lastly, the results and a detailed discussion of the experiments that prove the validity of the study, the usability of the proposed tool, and the improvements to the clustering methods are presented.

Keywords: Data distribution, Non-IID data, Data partitioning strategies, Algorithm evaluation, Benchmarking platform, Distributed data mining, Classification, Clustering

Streszczenie

Rozmiary różnych zbiorów danych gromadzonych na świecie gwałtownie rosną. Dane te są składowane w oddzielnych, niezależnych lokalizacjach. Z tego powodu wzrasta liczba nowych algorytmów do przetwarzania tak rozproszonych danych, w szczególności na potrzeby zadań klasyfikacji i grupowania. Nie ma jednak ustandaryzowanego sposobu walidacji takich algorytmów. Typowo są one testowane na niezależnych, lecz identycznie rozłożonych danych (IID), które są równomiernie rozproszone. Jednakże prawdziwy rozkład danych między niezależne węzły jest zazwyczaj nieznany, co wpływa na wyniki jak i samo przetwarzanie. Ta praca jest poświęcona poprawie oceny rozproszonych algorytmów przez zastosowanie nowych metod nierównomiernego (non-IID) partycjonowania danych. Ma to na celu złagodzenie wpływu niezdefiniowanego rozkładu danych na wyniki działania algorytmu przez ukazanie jego niedoskonałości. Ponadto w pracy zaproponowano standardowy zbiór nowych strategii partycjonowania do symulowania rozkładu non-IID na potrzeby ewaluacji algorytmów rozproszonych celem ujawnienia ukrytych niedoskonałości w ich przetwarzaniu i poprawienia jakości testowania.

Dodatkowo przedstawiona została nowa, prosta w użyciu i rozszerzalna platforma umożliwiająca dynamiczne dołączanie komponentów. W pracy zaadresowany został tym samym problem braku kompleksowych narzędzi do oceny (benchmarkingu) metod rozproszonej eksploracji danych (DDM) w rzeczywistym środowisku rozproszonym. Proponowana platforma umożliwia właściwą analizę wyników pod kątem wpływu na algorytm różnych aspektów przetwarzania takich jak końcowa jakość, obciążenie transferu i szczegółowe pomiary czasu w kolejnych etapach przetwarzania.

Poza tym w niniejszej pracy ujawniony został negatywny wpływ różnych rozkładów danych na jakość wyników przetwarzania algorytmów rozproszonych na przykładzie ewaluacji zestawu metod rozproszonej klasyfikacji i grupowania. Wprowadzono również zaadaptowany algorytm hybrydowy, który jest w stanie osiągnąć wysoką jakość niezależnie od specyficznego rozkładu danych i bez konieczności przyjmowania założeń o jednolitym rozkładzie przetwarzanych danych.

Na koniec przedstawione zostały wyniki wraz ze szczegółowym omówieniem eksperymentów, które potwierdzają zasadność badań, użyteczność proponowanego narzędzia oraz pokazują ulepszoną metodę grupowania rozproszonych danych.

Słowa kluczowe: Rozkład danych, Dane non-IID, Stragetie partycjonowania danych, Ewaluacja algorytmu, Platforma porównawcza, Rozproszona eksploracja danych, Klasyfikacja, Grupowanie

Contents

Chapter 1

Introduction

Classification, clustering and other data mining (DM) and machine learning (ML) methods have existed for a long time, and these terms are frequently used interchangeably. Many different approaches have emerged for processing the different types of data collected worldwide. Authors have usually verified the correctness of their algorithms by applying them to a small, local dataset, which typically then becomes a benchmarking set for similar new methods. The world has started to evolve rapidly, and massive amounts of data are now produced, which require changes in the approaches towards processing them. Initially, multiple parallel methods began to gain popularity; however, in the modern world, the data generated by organisations have started to be stored in independent remote storage, and privacy issues have arisen. Alongside optimisation, this requires the development of complete distributed algorithms that do not operate on the entire dataset, and has opened up a new area of application called distributed data mining (DDM). New challenges have been discovered that require new approaches and involve the need to overcome multiple limitations, such as the abovementioned privacy aspects, a lack of access to the whole dataset, increasing network transfer consumption and synchronisation issues. All of these challenges must be addressed when existing modern algorithms operate in distributed environments, and must also be considered in the design of new ones.

The continuous growth in the data collected worldwide accelerates every day, and has reached sizes on the order of zettabytes (ZB, or 10^{21} B) [13]. These sizes are predicted to reach between 163 [35] and 175 [3] ZB by 2025. Moreover, the countless devices that form the Internet of Things (IoT) also produce quintillions of bytes (10^{18} B) [18] of data every day. A total of 500 ZB of data was generated by such IoT sources in 2020 [54], due to their large numbers. However, a constantly growing number of new methods are still using a traditional approach to evaluation using the same existing datasets, with the strong assumption that the data characteristics will be equivalent for every independent storage method. Although an algorithm may achieve good results in a single setup scenario, authors typically do not investigate whether their algorithms will fail on multiple corner cases and thus will dramatically slow down in terms of processing or require that all the data are transferred to a single location for processing. These issues have initiated a strong interest in examining such algorithms, since non-uniform data distributions remain an open problem, especially since federated learning (FL) was introduced in 2017. The strong interest in FL arises from the data collected by the countless IoT devices. However, much of the data from these devices are typically gathered in data centres for 'data-centre computing', which is more closely associated with other DDM techniques in which the order of magnitude is smaller [12]. Data of this type are typically collected and stored in different geographic regions, and there is no guarantee of an equal distribution of data samples at each location. The data are then not independent and identically distributed (non-IID). These data are not only used for FL processing, but also other DDM techniques, and none of them should deteriorate results because of distribution characteristics. Finding a suitable dataset to evaluate a given method is also problematic. Nevertheless, one question that remains unanswered when a particular dataset is available is how to scatter the data to show that the algorithm works without a uniform data distribution. Another issue is that of how to perform a comprehensive evaluation.

1.1 Motivation

Various methods of data mining have been developed, and in the last few years, there has been growing interest in the study of FL techniques. In this approach, a large amount of data is distributed over huge numbers of devices that cooperate in the learning process. The concept of FL represents a particular case of general DDM methods, which also deal with distributed data but at different scales; for example, the processing of megabytes of data stored on millions of devices requires a different approach from the processing of petabytes of data stored in a few data centres. The DDM approach is not limited to the FL technique, and has been neglected in current research on the impact of data distributions and the evaluation of algorithms. We cannot forget the standard scale based on a low number of collaborating data centres and non-FL techniques of DDM, where the number of computing nodes is lower than in the collaboration between IoT devices. Moreover, most FL approaches in the literature are based on neural network architectures that are mainly dedicated to image classification tasks.

A lack of knowledge about the characteristics of the data distribution and an inability to download all of the data locally pose challenges for modern algorithms operating in distributed environments. No analyst can be certain of a complete data representation at every independent node in the real world, and there are countless possibilities for spreading an entire dataset among multiple nodes. A great deal of research has been done on simulating different data distributions to benchmark FL algorithms, mainly since the authors of [33] identified open problems associated with non-IID data distributions. The focus of recent research has primarily been on popular and available datasets as a fixed set of inputs for benchmarking. Methods of dividing a dataset into non-IID partitions are often briefly and vaguely described, with implementations that are rigidly dependent on the dataset, such as MNIST [40], CIFAR-10 [37] or Sentiment140 [20].

Such methods generally focus on image or textual datasets, which are currently of interest for FL, and neglect the more typical numerical and nominal data. New techniques of data partitioning for the evaluation of algorithms are required, but with approaches that are agnostic in terms of dataset type where possible. The need for standardisation is evident when preparing test suites, and an understanding is required of which distribution aspect is examined during experiments on data partitioning, especially in view of the lack of consensus on the metric of "non-IID-ness" for data [43, 28].

The dataset used to evaluate an algorithm is crucial. Several aspects of the evaluation process are important, such as how to perform it under similar conditions, collect results with numerous execution statistics and compare them. Multiple methods and approaches have been extensively used in recent studies; however, there is an ongoing discussion concerning how to verify the correctness of an algorithm in a standard way. There is a need to test a new approach against other methods under similar conditions. A typical algorithm evaluation is performed using a random, uniform data distribution or a cross-validation method. For strongly unbalanced sets, the number of class samples during scattering is usually also considered to simulate an even class distribution; however, in a completely distributed environment, the entire dataset cannot be accessed or is deliberately restricted for privacy reasons. In order to fully examine the algorithm, the algorithm should also be evaluated on a specific data distribution rather than in typical scenarios. As mentioned previously, every evaluation of a DDM algorithm involves an independent partitioned dataset, which implies communication between computational nodes. This takes time, consumes transfer bandwidth, and is a potential security issue when too many data are transferred. We have identified a gap in the area of tools for evaluating deterministic DDM algorithms, in terms of providing comprehensive and comparable results for various execution aspects. There is a lack of generic, easy-to-use tools that use different test suite parameterisations for data partitioning and consider the abovementioned issues related to time and transfer statistics analysis.

The provision of any platform for algorithm verification or benchmarking involves several challenges related to both flexibility and simplicity of use. Moreover, a major drawback in many studies is a lack of information about the similarity measures used in clustering and some classification algorithms, which makes it almost impossible to reproduce results on a specific dataset. Although it is important to allow developers and researchers to implement algorithms in their own way and to provide verification tools that require minimum effort to use, an algorithm still needs to follow certain rules when applied to enable execution and measurement, e.g., of the data loading time or the size of the transfer among worker nodes during execution. The quintessential example is the MapReduce [17] framework, in which the application has to implement both mapping and reducing functions. Secondly, the majority of the recent algorithms are not as generic as they were in the last century, and are designed to work with certain datasets. Many researchers still use data from the well-known UCI Machine Learning

Repository or KDD Cup 1999 Data for evaluation, and apply simple Euclidean similarity measures. However, an increasing number of new classifiers have been dedicated, e.g., to financial datasets for fraud detection [1] or to clustering methods used for the selection of reference sets from genomic data [39], and so on. As a result, it is essential for a benchmarking platform to handle the comparison of different data objects for algorithm execution and data scattering among nodes in a non-uniform manner using data partitioning strategies. An easy way of using custom similarity measures between samples must be available in the system. The way in which data are distributed in a distributed processing environment dictates the execution performance and the quality of the algorithm.

1.2 Thesis and goals

1.2.1 Goals

The main goal of this work is to develop various methods of simulating different non-IID data distributions, which can affect the deterioration of distributed clustering results and the quality of classification algorithms. A further objective is to design and implement a fast, reliable algorithm that does not require any assumptions regarding the data distribution. In addition, an evaluation platform will be implemented to conduct reproducible experiments efficiently using various datasets, data partitioning and algorithms to verify the influence of the proposed methods. The results will be analysed in terms of the impact of the data partitioning strategies on the quality of the results, particularly with regard to partitioning-type clustering methods and SVM classification.

1.2.2 Thesis

The various strategies that can be used for data dispersion among independent computational nodes strongly impact the results of distributed clustering and classification algorithms. Current methods of examining distributed algorithms that operate on independent datasets are incomplete, and require consideration of the impact of uneven data distributions on the operation of the algorithm. It is possible to define an initial assessment of the scale of such impact of the partitioning strategy on the type of algorithm.

The proposed data partitioning methods, an analysis of their influence and the proposed evaluation approach extend the possibilities of verifying the quality of an algorithm, and include the following:

- A set of methods to simulate uneven dataset dispersion among independent computational nodes, which allow for an examination of the correctness of the algorithm's operation without requiring assumptions about a uniform data distribution;

- A reference hybrid clustering algorithm that is adapted to work with uneven data distributions;
- A tool dedicated to the evaluation of distributed algorithms with non-IID data distributions;
- An identified scale of the impact of particular data distribution on different algorithm groups.

1.3 Scope and limits of this work

The areas of data mining and distributed processing involve various methods, approaches, data and applications. The range of possibilities is further increased by modern methods that are designed to work in distributed environments by mixing different approaches, thus creating hybrid algorithms. Many surveys have dealt with this constantly growing variety of methods and tools, and have described these topics in a few sentences. In view of this fast-growing diversity, and to develop a more profound insight, we focus on a set of methods including the probabilistic and SVM classifiers, and partitioning clustering algorithms enhanced by density and hierarchical approaches. Studies that mention non-uniform data distributions typically consider neural network classifier applications, which are omitted from this work, although these are mentioned in the review of data partitioning. The scope of this study is also limited to horizontal partitioning, as the more common method, and numerical and nominal data, although other types will be mentioned as they occur in related work.

1.4 Structure

This work is structured as follows. In the next section, we introduce basic information related to the topic, including data partitioning and similarity, types of algorithm and distributed data processing. In Section 3, current work on benchmarking of the distributed algorithms is summarised, and an overview of different data distribution categories is provided. The proposed evaluation method, including new data partitioning methods, the algorithm and the platform, is described in Section 4. Section 5 reports the results of experiments based on our evaluation process, and presents a discussion. Finally, Section 6 concludes the thesis and suggests directions for future work.

Chapter 2

Foundations

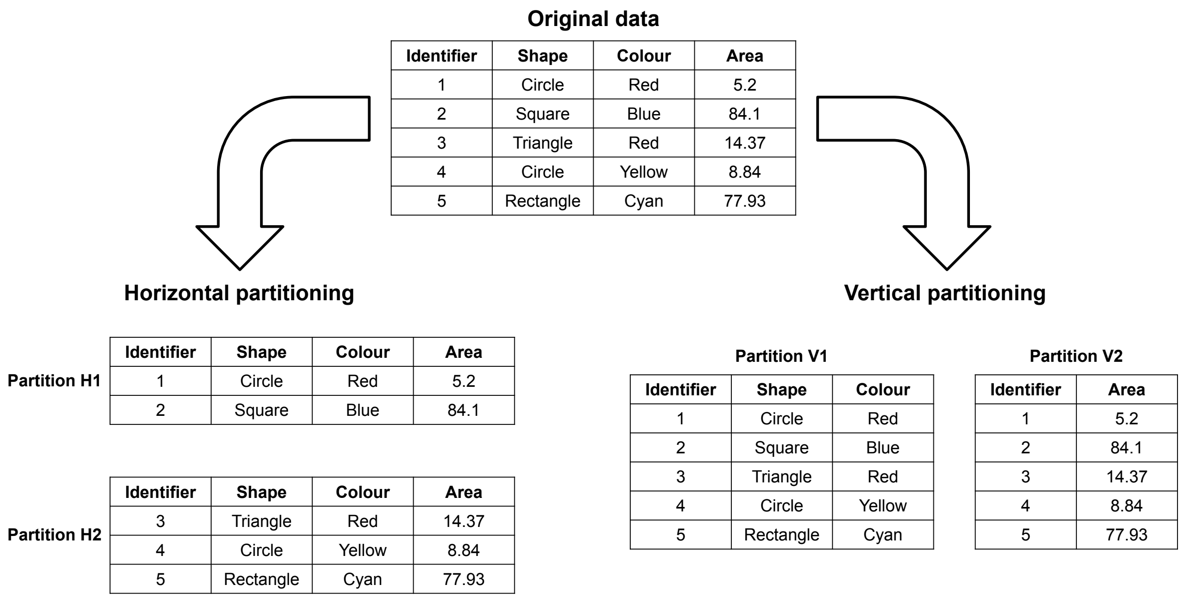
2.1 Types of data

We can distinguish between many types of data that are collected and stored across the world. Such data consist of observations or measurements, which are represented as numbers and strings. The most basic distinction is between numerical and nominal (categorical) data. The ordinal type can also be distinguished as nominal data with a defined order.

We can also distinguish textual datasets, consisting of structured characters or entire documents, as a distinct data type. However, this is a rather specific data structure that is created to represent some knowledge or information. Other types of data are derived from the main groups but involve certain semantics, or follow predefined structures. Various other types of data exist, such as sequential, image, or domain-specific data. For instance, time-series data may have any type, and can represent anything marked with a timestamp which defines the occurrence of each entry over time. In contrast, an image representation is limited by a finite set of numbers in two-dimensional space. Domain-specific data depend on the problem; for instance, genomic data involve long structures consisting of specific and rare sets of values, and are both nominal and sequential.

2.2 Data partitioning

Datasets can be divided into two primary ways. The most common is horizontal partitioning, although many types of structured data can also be partitioned vertically. These approaches are illustrated in Fig. 2.1. In the first type, each partition contains all of the attributes describing the data entries, but not the whole dataset. In contrast, vertical partitioning involves the splitting of the attributes of the whole dataset between the partitions. The most common application is in relational databases and table normalisation, but this approach is not limited to this type of issue. The choice of a data partitioning method depends on the problem to be solved, and defining a data model depends on the requirements and read/write optimisation. Some types of data partitioning in real production systems may arise spontaneously across independent data storage centres; for example, structured data collected in different but related data centres can be naturally split horizontally.



Original data

Identifier	Shape	Colour	Area
1	Circle	Red	5.2
2	Square	Blue	84.1
3	Triangle	Red	14.37
4	Circle	Yellow	8.84
5	Rectangle	Cyan	77.93

Partition H1

Identifier	Shape	Colour	Area
1	Circle	Red	5.2
2	Square	Blue	84.1

Partition H2

Identifier	Shape	Colour	Area
3	Triangle	Red	14.37
4	Circle	Yellow	8.84
5	Rectangle	Cyan	77.93

Partition V1

Identifier	Shape	Colour
1	Circle	Red
2	Square	Blue
3	Triangle	Red
4	Circle	Yellow
5	Rectangle	Cyan

Partition V2

Identifier	Area
1	5.2
2	84.1
3	14.37
4	8.84
5	77.93

Figure 2.1: Examples of horizontal and vertical data partitioning.

Horizontal partitioning may vary across partitions, depending on the characteristics of the stored data. For instance, in relational databases, rows can be partitioned with different criteria based on ranges, hashes or round-robin strategies. This type of division involves intentional placement, whereas the collected data may be naturally characterised by different distributions in the distributed independent storage centres.

2.2.1 IID data distribution

Data located on separated partitions and sharing equivalent distribution characteristics of their attributes are known as independent and identically distributed (IID). A dataset distributed across partitions is referred to as IID when the data samples are uniformly spread over the partitions; in other words, if each data sample with certain characteristics or attributes can be found on any partition with the same mutual independent probability distribution, then the whole dataset is IID.

The original formalism comes from statistics based on the definition of the independence of random variables. We assume that a data sample from a dataset D can be considered as a realisation of a multidimensional random variable x. We also assume n separated partitions and the function $F_{X_k}(x)$, defined as the probability of finding sample x on partition k. We can then formally define the following dependence for a dataset distribution which is IID:

Definition 2.2.1. The distribution of dataset $X = \{X_1, \ldots, X_k\}$ is independent and identical if and only if:

$$\forall i, j \in \{1, \ldots, n\},\ F_{X_i}(x) = F_{X_j}(x) \tag{2.1}$$

and is mutually independent if:

$$F_{X_1,\ldots,X_n}(x_1, \ldots, x_n) = F_{X_1}(x_1) \cdot \ldots \cdot F_{X_n}(x_n) \tag{2.2}$$

2.2.2 Non-IID data distribution

Data can be unbalanced between partitions, and thus located non-uniformly, in many ways. Referring to the definition of IID data, we can define a non-uniform data distribution as the negation of Definition 2.2.1. When data are non-ideal or non-identically distributed, regardless of the cause, they are called non-IID data. According to the taxonomy of non-IID data regimes [33], we can distinguish five main categories of possible real-world distributions for any partitioned dataset:

- **Feature distribution skew** (covariate shift): This category refers to the situation where different attributes are distributed on separate partitions. Examples include typical differences in handwriting characteristics, speed limit signs that differ slightly between European countries [21], or attributes describing the same plant species based on their size, colour, etc., which vary depending on the weather conditions and sunlight exposure;

- **Label distribution skew** (prior probability shift): This relates to the distribution of data samples in object classes present on different partitions. This category is probably the most likely, especially when processing data from different geographic locations. A typical example of this would be the presence of kangaroos in various regions around the world: this animal lives in the wild only in Australia, although single individuals can be found in a few zoos around the world. Another situation is represented by people from different countries who emigrate and settle in groups, which affect the distribution of local communities. A third situation is the use of English in books or conversations, which is of course mainly found in the US or UK, although this particular language is also used all over the world. These examples involve numerous label-skew combinations depending on the presence and counts of data samples on different partitions. We can distinguish between them further for the purposes of partitioning;

- **Same label, different features** (concept drift): This distribution category describes a situation where possibly non-intersecting attribute values of data samples located on separate partitions represent the same class of objects. The most common example is the classification of pictures of houses, where photographs of houses are taken at different times of the year or under different weather conditions. Another example is the features of people from different regions of the world who differ in terms of their height, weight, hair colour, etc., which has an impact when exploring data on medical conditions;

- **Same features, different label** (concept shift): This refers to the situation where data are understood differently or are misclassified on separate partitions with data samples representing the same object class. The concept underlying this issue relates to divergence in the class identification of data objects described by the same features in different regions, or simply a different understanding of what is represented by the characteristics when marking the data used for training. This is closely related to the natural language processing (NLP) process for sentiment analysis problems, where words can be tagged in different ways in the training dataset depending on the understanding. An example of nominal data might be a different understanding of what constitutes a powerful or wealthy person in different countries;

- **Quantity skew** (unbalancedness): This is where data are highly unbalanced across partitions in terms of sample quantity. In this case, we assume that some nodes have more data samples, and others have significantly fewer.

2.3 Data mining algorithms

To extract information or find knowledge from data, we need to process them by applying dedicated algorithms. There are many types of algorithms that were developed for different purposes, and the basic groups are as follows: classification, clustering, regression or prediction and association rules. All of these are included in the general concept of data mining and ML methods. Moreover, these methods can also be classified at various levels due to the approach used, the results produced or even the models used in processing. An approach may involve a distinction between supervised or unsupervised learning methods. The results produced by these algorithms may be precise or fuzzy.

Most types of algorithm create a model to solve a particular problem using different techniques. For instance, classification tasks can be solved using either decision trees or neural networks, and the models used in these cases are entirely different. These methods, however, are not limited in application to this single task.

Modern techniques also combine core algorithms to achieve better results and to exploit the advantages of each one. This type of combination is called an ensemble method or hybrid algorithm, in which two or more existing algorithms are combined. The present work focuses on clustering and classification tasks in the processing of distributed data.

2.3.1 Clustering

Clustering is a technique for grouping similar objects together based on an arbitrarily chosen measure. The similarity between two objects depends on the objective and the data, and similarity measures are discussed in more detail in the following section. We can divide clustering algorithms into five main types or categories (depending on the nomenclature), as recently summarised in [19] and extensively described in [59]. Each describes the approach to processing taken by the algorithms to perform final clustering. We distinguish the following clustering types:

- **Partitioning**: This type of algorithm divides the space of the processed dataset into a predefined number of partitions. Each partition represents the final cluster, consisting of data within the partition boundary. Iterative relocation is performed by moving data from one cluster to another, starting from an initial partitioning. The most well-known type of partitioning clustering is the k-means algorithm [23], which is illustrated with an example in Fig. 2.4 below;

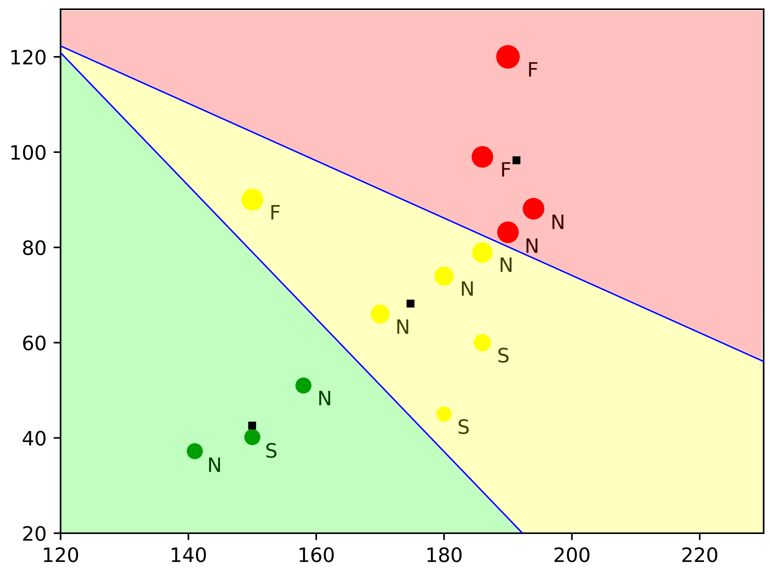


Figure 2.2: Example of partitioning in Euclidean space by a partitioning clustering method (colours define the found clusters, with black dots as centres; letters indicate the reference object class, which was ignored during unsupervised learning).

- **Hierarchical**: A data cluster representation is organised in the form of a dendrogram, with objects as leaves. This type of clustering may use two methods of building a clustering structure, depending on the starting point: agglomerative (bottom-up) or divisive (top-down). The former starts with the processing of clusters containing single objects and then iteratively merges them based on mutual distance, while the latter performs the reverse operations, and the whole dataset is split into smaller clusters. Both algorithms stop processing when they reach a defined stopping criterion, usually based on a particular number of found clusters. Using this type of approach, we can obtain different clusters, even consisting of nested groups, by changing the stopping criterion for processing. An example is presented in Fig. 2.3. Algorithms of this type include BIRCH [71] and CURE [22];

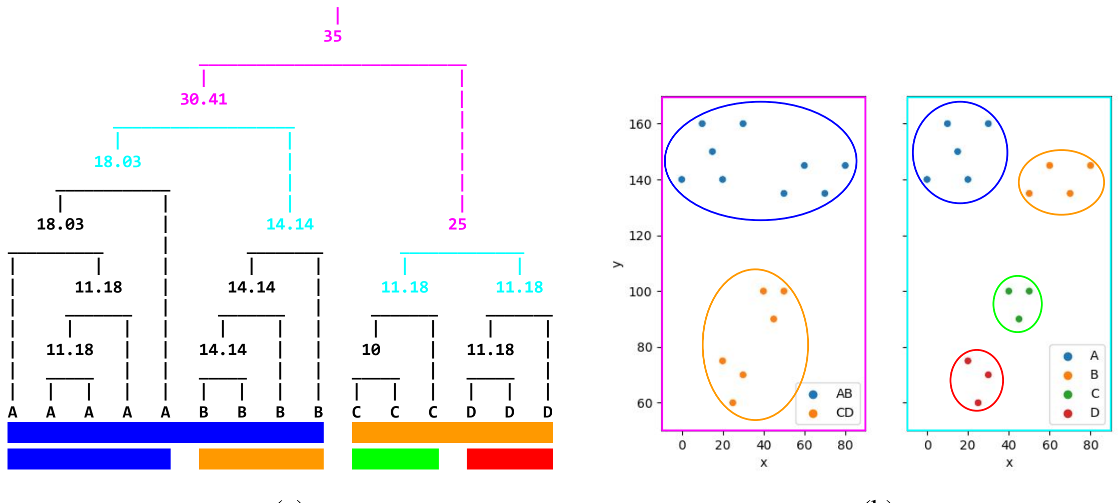


Figure 2.3: (a) Example dendrogram showing the hierarchical clustering method; (b) changes in clustering results depending on the stop criterion (two or four groups) for processing data.

- **Density**: In this method, data objects are grouped into clusters based on the connectivity between the objects. Depending on the parameterisation for the chosen algorithm, data samples take the form of dense groups in terms of the specified distance measure. Processing usually starts with a single object and grows until the boundary is reached, defined as a non-connective neighbour. In the density-based approach, algorithms naturally find outliers, anomalies or simply noise in data. An example of the processing results of the OPTICS [4] algorithm depending on the *'density'* parameterisation is shown in Fig. 2.4;

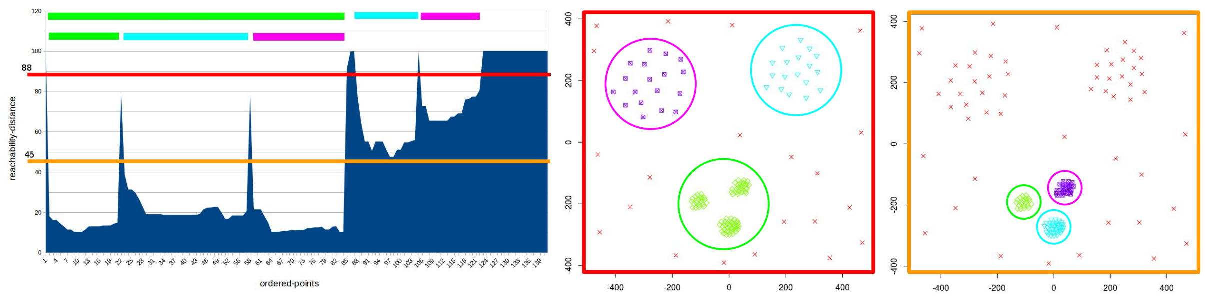


Figure 2.4: (a) Example of the resulting structure of the OPTICS algorithm as ordered points and their neighbour reachability distance; (b) differences in the possible clustering results depending on the ε' parameter.

- **Grid**: This method divides the data space into a grid with a specified size. Each grid cell is then processed by analysing the density of the objects present within it, and by combining neighbour cells into clusters when the neighbour density reaches a specified threshold. This behaviour is similar to the agglomerative approach of a hierarchical clustering algorithm. The processing time and complexity depend more on the size of the grid than the size of the dataset;

- **Model**: In this type, we assign data objects into groups based on a predefined mathematical function. The assumption is that based on the statistics of the data, we can define a function that describes the data characteristics and hence form a clustering model. The algorithm then tries to optimise the assignment of data into clusters by fitting the data to the defined model.

The complexity, parameterisation, and types of data that an algorithm can process depend on the particular implementation. In general, assigning an algorithm to a particular type is an arbitrary decision based on an analysis of how the algorithm processes data samples to form groups.

The range of applications of clustering is wide, from exploring data or identifying patterns using cluster analysis to anomaly detection. The choice of a suitable algorithm for a specific task depends on the problem, and requires *a priori* knowledge of the data characteristics and volume.

2.3.2 Classification

Unlike the clustering task, in which learning is unsupervised, classification is a supervised learning method. The aim is to determine or identify an observation category based on the model. The model is built based on the training set, and *a priori* knowledge of the data in this set is required, meaning that classifiers cannot categorise observations into previously unknown classes. Nevertheless, depending on the classification model, we can often state the confidence level of the predicted class.

We cannot divide classification methods into a fixed number of types in the same way as clustering methods, as classifiers differ based on the approach used to build and apply a model. We can, however, list the most popular approaches and methods as follows:

- **Probabilistic**: In this approach, prediction is based on a statistical probability analysis of objects and their attributes, where the probability defines the possibility of being categorised into each class. The most well-known example of such a classifier is naive Bayes [65];

- **Neighbour**: Here, assignment of a new observation is made based on its similarity to the closest object in the training set, taking its class as the prediction. The k-NN classifier [55] is a prominent example, as it is probably the most widely used in practice due to its simplicity;

- **Decision trees**: In this method, prediction is defined by a set of decisions based on observation attributes that lead to the leaf representing the target class. The structure of the model is in the form of a tree, with branches describing decisions and leaves corresponding to the final categorisation. A sequence of decisions from the root defines a path to the prediction. This is probably the most popular method, with various implementations, and is easily applicable to both numerical and nominal data;
- **Support vector machines** (SVMs): This is a family of classifiers in which data are divided into two categories in the training phase, falling 'above' and 'below' an n-dimensional plane defined as a function based on the boundary points (support vectors) and kernel function used. The algorithm tries to maximise the gap between two classes of objects by fitting the kernel function and thus finding support vectors. The great advantage of this method is the possibility of changing the kernel function to classify well linearly separable data and perform nonlinear separation. This is an example of a binary classifier, and applying it to a multiclass problem requires additional actions that are described later in this section;
- **Boosting** (ensemble): This method assumes the utilisation of a meta-algorithm based on multiple weak classifiers that finally create a single strong classifier. Weak classifiers are defined as those that are not well correlated to the problem but are better than random guessing. For instance, a single weak classifier can be trained on a subset of the training data. Multiple methods use this approach, often based on decision trees such as the random forest or AdaBoost algorithms;
- **Neural networks**: These are probably the most popular classification method nowadays; they were inspired by biological neural networks, and provide outstanding prediction results. The architecture of the network depends on the problem to be solved in terms of the number of inputs (usually data attributes) and outputs (classes). A typical architecture is based on a graph structure consisting of a few layers and initial weights assigned to nodes. The nodes between the input and output nodes of the graph are the layers. In the learning process, the weights are updated based on the training samples and the learning function, allowing the final model to predict new observations correctly. Numerous different types and architectures of neural networks exist in the literature, and they are widely studied, although they fall outside the scope of this work.

In addition, we distinguish two types of classification: binary and multiclass. In the former, the classification result is binary; for instance, a spam filtering algorithm may determine whether a message is wanted or not. Many classifiers are designed for binary classification, such as the SVM classifier, although the multiclass problem can be handled by splitting the dataset into multiple binary classification datasets. This can be done using two strategies. The first, which is known as the one-vs-rest (OvR) strategy, assumes the creation of multiple datasets for comparison of each class with the remainder. In the second, which is called the one-vs-one (OvO) strategy, the created datasets consist of each pair of classes for comparison. Regardless of the chosen strategy, we can build multiple binary classifiers, the number of which depends on the number of classes available in the dataset. For the OvR strategy, we build N classifiers for N distinct classes, while for the OvO strategy, we need to prepare $\frac{N\cdot(N-1)}{2}$ binary classifiers. These binary classifiers produce predictions which differ from one another, and we therefore need to decide how to choose the final result using a voting strategy. The range of possible voting strategies is large [72], but the most typical is majority voting, in which the most frequent prediction is chosen as the final result.

2.4 Data similarity

Any algorithm that processes data for a clustering or classification task needs to compare data samples to each other, and a similarity measure is a way of enabling this comparison. We can also distinguish a dissimilarity measure, also known as a distance measure, which defines how distinct two data objects are. Similarity is usually expressed in numerical values in the range $[0,1]$, or as a binary quantity for nominal attributes; typically, 0 means no similarity at all, while 1 means indistinguishability, although this is not a rule. In some cases, we may not know the maximum value of the dissimilarity between some data, for example in numerical space, and we then use the 'distance' measure instead. The relationship between similarity and distance is such that a smaller distance indicates a higher similarity between the data.

A similarity measure is typically used in conjunction with classification tasks, in which the aim is to predict the class by finding the similarity in the model. The model is trained on data that describe new potential observations, so we expect similarity. In the clustering domain, the term 'distance measure' is often used interchangeably with 'similarity'. This is because we group similar objects together and divide data into separated clusters so that objects from various clusters show high dissimilarity.

There are many similarity measures that have been defined, and the most suitable choice depends on the data being processed and the problem to be solved or the question to be answered. A broad survey of existing measures can be found in [14]. Commonly, algorithms operating on numerical data use the Euclidean distance to compare data samples; however, this is not a rule. The cosine measure works better when comparing vector representations of text documents. On the other hand, specialised algorithms can work with custom similarity metrics; a comparison of genomic data, described by specialised structures, is an excellent example.

2.4.1 Metrics

A distance measure can be considered a metric when it meets several conditions. Not every measure can meet these conditions, but if an algorithm knows how to utilise metrics features for processing, this can have a strong impact in terms of better performance. For instance, the results obtained by the authors of [38] suggest that using the triangle inequality property to reduce the neighbourhood search space makes the improved DBSCAN clustering algorithm much more efficient, especially for high-dimensional data. Nevertheless, a given measure can be acknowledged as a metric if and only if it satisfies the following four conditions:

1. Non-negativity: $d(p,q) \geq 0$ for any two distinct observations p and q,
2. Symmetry: $d(p,q) = d(q,p)$ for all p and q,
3. Triangle inequality: $d(p,q) \leq d(p,r) + d(r,q)$ for all p, q, r,
4. $d(p,q) = 0$ only if $p = q$.

The Euclidean distance, used in the work mentioned above, is a metric that is defined as follows:

Definition 2.4.1. The Euclidean distance for points given by Cartesian coordinates in n-dimensional Euclidean space is defined as:

$$d(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \ldots + (p_n - q_n)^2} \tag{2.3}$$

In contrast, the cosine distance is not a metric, as it does not satisfy the fourth condition and the triangle inequality. A cosine similarity and cosine distance are defined as follows:

Definition 2.4.2. The cosine similarity measures the angle between two vectors projected in a multi-dimensional space, and is expressed as:

$$S_C(A,B) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum\limits_{i=1}^{n} A_i B_i}{\sqrt{\sum\limits_{i=1}^{n} A_i^2}\sqrt{\sum\limits_{i=1}^{n} B_i^2}} \tag{2.4}$$

The result value ranges from -1 to 1. The lower limit represents exactly opposite objects, while the higher limit represents fully similar objects. 0 indicates a lack of correlation between the objects.

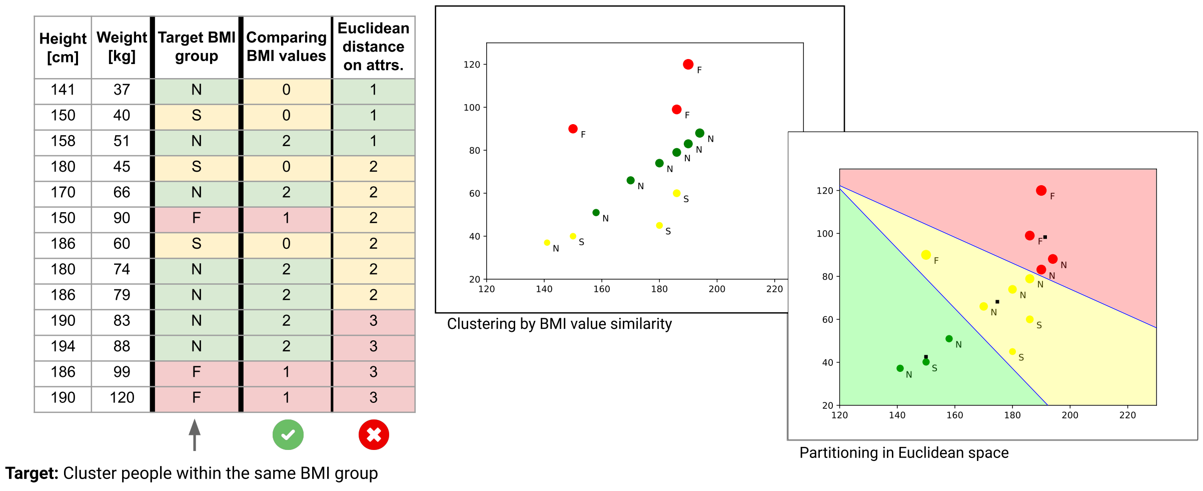
Definition 2.4.3. The cosine distance is commonly used as a complement of cosine similarity in positive space, and is expressed as:

$$D_C(A,B) = 1 - S_C(A,B) \tag{2.5}$$

The possible ranges of the result are values shifted to positive space that starts from 0 and ends with 2, where the dissimilarity of the objects increases together with higher formula results.

2.4.2 Custom measures

Depending on the application, a single metric may be most suitable; however, processed data sometimes require custom metrics, or lean on specific search goals. Fig. 2.5 gives an example of the different results obtained using different measures for the k-means clustering algorithm. When we compare the data using the pure Euclidean distance, we obtain some results, but short and overweight people are grouped together with average ones. To group people based on their body structure, we could use a custom measure based on the Euclidean distance but in one dimension, by comparing calculated BMI indices. Of course, these results could be obtained with simple calculations, but this example illustrates the differences in the results that can be obtained and the possibility of applying custom measures.



Height [cm]	Weight [kg]	Target BMI group	Comparing BMI values	Euclidean distance on attrs.
141	37	N	0	1
150	40	S	0	1
158	51	N	2	1
180	45	S	0	2
170	66	N	2	2
150	90	F	1	2
186	60	S	0	2
180	74	N	2	2
186	79	N	2	2
190	83	N	2	3
194	88	N	2	3
186	99	F	1	3
190	120	F	1	3
		↑	✓	✗

Target: Cluster people within the same BMI group

Figure 2.5: Example of the way in which different similarity or distance metrics may affect the clustering results.

2.5 Measures of the quality of results

Calculating the quality of the results in order to evaluate an algorithm depends on the data mining method used, the data and their characteristics. A suitable metric selection will ensure correct results when new data are processed. Evaluation criteria are typically divided into two main categories: external and internal.

The internal criterion is applied to unsupervised clustering methods. This kind of criterion is used to analyse the final clusters, typically by calculating their compactness, homogeneity and separability. No external information is used in this process besides the data themselves and the cluster assignments. Various internal validity measures exist in the literature [59, 47, 64], depending on the requirements and the characteristics of the data. Nevertheless, when we evaluate an algorithm, we typically expect certain results, and can therefore use an external measure as a first choice. In addition, obtaining more information about the characteristics of the results and the data can be helpful.

The external category is based on reference data results that are known in advance. For a clustering task, this is usually a reference clustering for the evaluated dataset. To build a classifier, a labelled training dataset is required, with a separate test set for verification. In classification, the training set cannot be used as the test set, because we would then be testing the algorithm on already known samples, which is likely to produce perfect results while the model may fail on new observations. The test set should therefore be a separately provided dataset or subset extracted for testing purposes. Other possibilities involve the use of cross-validation methods for evaluation. After obtaining predictions for the testing set or assignments to clusters, we need to compare them with references. Several metrics have been proposed, as described in [59] for clustering algorithms and in [25] for classification algorithms.

2.5.1 Clustering evaluation metrics

The most popular external metrics are the Rand index (RI) and mutual information (MI). RI is a simple metric that considers all pairs of data by counting their assignment to the same or different clusters, and then compares the totals of the two clustering results. Clustering algorithms may assign classes to data samples in an undefined order, but this does not impact the index value. The final index lies in the range $[0,1]$, where the highest value indicates a perfect match. Let C_1 be the reference clustering and C_2 be the obtained results. RI is then defined as shown in Equation 2.6:

$$RI = \frac{a+b}{a+b+c+d} \equiv \frac{number\ of\ matched\ pairs}{number\ of\ pairs} \tag{2.6}$$

where a is the number of pairs of data assigned to the same cluster in both C_1 and C_2, b is the number of pairs in the same cluster in C_1 but in a different cluster in C_2, c is the number of pairs in the same cluster in C_2 but in a different cluster in C_1, and d is the number of pairs assigned to different clusters in C_1 and C_2.

In the basic version, a pair of totally disjoint results does not produce the lowest value of 0 for the RI. The remedy for this is the adjusted Rand index (ARI), introduced in [29], which can be written in terms of the RI as shown in Equation 2.7:

$$ARI = \frac{\sum_{ij}\binom{n_{ij}}{2} - \left[\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}\right] \Big/ \binom{n}{2}}{\frac{1}{2}\left[\sum_i \binom{a_i}{2} + \sum_j \binom{b_j}{2}\right] - \left[\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}\right] \Big/ \binom{n}{2}} \equiv \frac{RI - ExpectedRI}{Max(RI) - ExpectedRI} \tag{2.7}$$

MI metric measures mutual dependence between two random variables that may also be discrete by representing clustering results. Intuitively, it shows the reduction in uncertainty about one random variable by knowing another. The value is always greater than or equal to 0, where a higher value indicates a higher correlation between two variables. Variables are independent when the value is 0, which indicates no matching clusters when clustering results are compared. The MI score also has an adjusted version that does not overestimate the results for two clustering results when a large number of clusters are found. The basic version is represented by Equation 2.8 in the normalised form, to fit the range $[0,1]$, while the adjusted mutual information (AMI) version can be expressed similarly to the ARI except for the replacement of *RI* with *MI* in Equation 2.7. Calculation of this function, however, is significantly slower to compute than for the ARI.

$$MI(U,V) = \sum_{i=1}^{|U|} \sum_{j=1}^{|V|} \frac{|U_i \cap V_j|}{N} \log \frac{N|U_i \cap V_j|}{|U_i||V_j|} \tag{2.8}$$

where by $|X|$ we denote the cardinality of the set X, U_i are the reference clusters, V_j are the resulting clusters for the evaluation, and N is the total number of samples.

The selection of a measure that will give meaningful results depends on the data; for instance, more reliable results are provided using the ARI for large, equally sized clusters, while the use of AMI is better when small, unbalanced clusters exist in the data [60].

2.5.2 Classification evaluation metrics

Several metrics were summarised in [25]. In practice, the most common of these, due to the information that can be provided, are the accuracy, F1-score (also called F-measure), and the precision and recall, which are components of the F1-score. In the literature, weighted versions of most metrics exist that are related to the numbers of samples in the data; although these can be helpful, we omit them here for simplicity.

The metric of accuracy measures the ratio of correct predictions to the total number of instances processed. It is defined as shown in Equation 2.9 for binary classification tasks, while Equation 2.10 defines it for a multiclass problem.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.9}$$

where the components refer to the standard classification-related terms: true positive, false positive, true negative and false negative.

$$Accuracy_{multiclass} = \frac{\sum_i^{|C|} ACC_{Ci}}{N} \tag{2.10}$$

where ACC_{C_i} is the accuracy measure for a single class C_i, $|C|$ is the number of classes, and N is the total number of predictions.

Precision and recall are variations of the accuracy metric that can answer different questions: precision gives information on how accurately we can predict positive patterns, while recall provides information about the proportion of positive patterns in the correct predictions. These are defined in Equations 2.11 and 2.12. Using these expressions, we can define the formula for F1-score as in Equation 2.13, which represents a harmonic mean between the values of precision and recall.

$$Precision = \frac{TP}{TP+FP} \tag{2.11}$$

$$Recall = \frac{TP}{TP+FN} \tag{2.12}$$

$$F1\text{-}score = 2 \cdot \frac{Precision \cdot Recall}{Pprecision + Recall} \tag{2.13}$$

As with clustering, the same choice of appropriate measure selection applies to the classification task, which also depends on data. For example, the F1-score gives more comprehensive information when processing highly unbalanced datasets [30].

2.6 Distributed processing

In the past, standard algorithms have been designed to work sequentially, with access to the whole dataset at once. However, we cannot simply run an algorithm such as this in a parallel or distributed mode due to the implementation, which is unsuitable for this purpose. Hence, new algorithms have begun to be developed, initially for parallel operation and then for a distributed environment.

In distributed processing, parallelisation may be applied within local computation units in order to speed up algorithms. Explicit differences between parallel and distributed processing are often subtle, and should therefore be clarified. In distributed processing, computations are run simultaneously on multiple independent nodes and datasets, while the parallel approach is based on access to the entire dataset at once.

The essential aspect of both parallel or distributed processing is the orchestration of the execution steps and access to the data resources. The data may be located in a single storage centre and accessed by all the computational nodes; a more complicated situation that needs to be handled by the algorithm is when data are stored in independent locations, and only certain computational units have access. Various architectures have been developed for distributed data processing. The general term for the group of algorithms and tools that "mine" data in a distributed environment is DDM. Two main concepts underlying fully distributed computation can be identified in the literature. Originally, the most commonly used approach relied on a central coordinator [31, 32, 57] for communication. A recent, widely studied concept is decentralisation utilising peer-to-peer communication between nodes [7, 16, 58]. Since 2017, however, the most popular and extensively developed approach has begun to be FL, which was introduced by Google and returned to the use of a central coordinator. Each of these concepts has specific advantages and disadvantages; in particular, a peer-to-peer (P2P) architecture requires a more complex infrastructure and more sophisticated methods of communication.

It is also worth mentioning that each type of distributed processing in data mining is a specific technique of the more general DDM approach. Different terminologies are used, and multiple tools have been developed, but the main goal remains the same: to process data in a distributed manner.

2.6.1 Federated Learning

FL is a technique for ML in which multiple independent parties are involved in the learning process. This technique represents a particular case of DDM, and is also known as collaborative learning. In 2017, research scientists Brendan McMahan and Daniel Ramage introduced this approach in a blog post[1] entitled "Federated Learning: Collaborative Machine Learning without Centralized Training Data".

In this processing model, the whole dataset is not collected in one central place, but is kept on local devices or nodes. The primary goal of this technique is to make the model smarter by using data from multiple independent devices, thus avoiding the transfer of local data to the central site and reducing latency through the use of small updates. The learning process starts with the initial model sent by the central server to every worker node. Next, each worker trains his model using its local data and returns the difference. The central node collects these differences and performs averaging to create a global prediction model. The process can then be iterated, depending on the algorithm chosen.

2.6.2 Spark

A large number of algorithms have focused on the current leading distributed processing platform, which is Spark [70] working on a Hadoop [63] cluster. The Spark platform provides an abstraction of the data called RDD, which represents a read-only multiset of the data and an interface for processing large amounts of data in a distributed environment, consisting of multiple workers in a cluster. The platform manages the cluster and sends the code to the workers for execution, thereby minimising data transfers and the number of execution steps required to finish processing. Following years of development, the current version of the platform has been extended, and now offers multiple features, from the execution of ML/big data algorithms to the application of SQL queries to big data and support for streaming algorithms. However, the essential feature (and the main drawback) is that it seamlessly handles both the orchestration of execution and data access. The drawback lies in the fact that the Spark platform controls the execution and the data source in an unspecified—and, more importantly, uncontrolled—way. Security awareness is high in modern distributed collaborations between institutions that rely on the processing of distributed data, and the privacy aspects of this unknown behaviour are potential grounds for exclusion of this platform from collaborations outside the industry.

[1]https://ai.googleblog.com/2017/04/federated-learning-collaborative.html

2.6.3 Docker

Although Docker [53] is not a tool or framework that is directly applicable for distributed processing, it is widely used for both simulation and application scalability purposes. This tool allows for virtualisation at the operating system level, creating isolated containers that can communicate with external entities when configured. The Docker system is commonly used for production applications and development, as it makes it easy to set up and clean the entire environment. It is extensively used for real distributed processing due to its simplicity and the high level of control that is possible. The most important aspect in terms of security and control is that by using Docker, we can easily separate containers from other systems and control access to them and the available resources.

Chapter 3

Related work and methods

The process used to evaluate different algorithmic methods depends on the purpose of the algorithm, and involves different execution aspects and evaluation approaches. The usual issues that are analysed include how well the algorithm performs in terms of its quality, how fast it runs, and the network load and communication. We can identify two main trends in the evaluation of these algorithms: the first is related to the data, and is more theoretical and formal, with a focus on distribution characteristics, while the second relates to benchmarking algorithms for orchestration and communication alongside performance measurements.

In the literature, there is substantial diversity in the use of terminology, and the term 'framework' is used interchangeably with 'platform' and even 'testing methodology' is confused with 'benchmarking'. In fact, their meanings are not identical. This section divides approaches to evaluation into categories based on their origin, type and purpose, and places them in context with regard to the main trends. We first describe the categorisation of the data distribution problems faced by modern distributed algorithms when working with independent data nodes with different distributions, usually with limited access.

3.1 Categorisation of non-IID data distributions

In a paper from 2019 on the open problems in FL [33], the authors introduced a taxonomy of non-IID data regimes, particularly for FL, and identified the main categories for one such regime. A visualisation of this taxonomy is presented in Fig. 4.1. We recall that the recently developed and well-known FL is a specific technique of the more general DDM approach, where the abovementioned open issues in the processing distributed data apply regardless of the technique used. The authors do not refer to a specific data type, but rather to the characteristics of data attributes that may vary for different client nodes holding data. They defined five categories of "non-identical client" distribution that closely reflect the possible data distributions in the real world, and examples of these are given in Section 2.2.2. However, a single category of data distribution includes various forms of dataset partitioning. Based on the simplest category of 'unbalancedness', an imbalance can be expressed as an inequality such as $A(p) < A(q)$ or

$A(p) \ll A(q)$, where A describes the amounts of data on partitions p and q. This simple example shows missing subcategories in the taxonomy required to distinguish diverse forms of data partitioning under the main categories. In this work, we give an overview of partitioning strategies and provide a more detailed analysis of this taxonomy in Section 4.1.1, where an extended version of this taxonomy is proposed.

Another categorisation scheme for non-IID data, with a strong focus on images and time series, which are partially beyond the scope of this work, is presented in [73]. The authors do not use the term 'taxonomy', and instead refer to a 'categorisation'. These categories can be presented in a tree-like hierarchy with subcategories, as illustrated in Fig. 3.1.

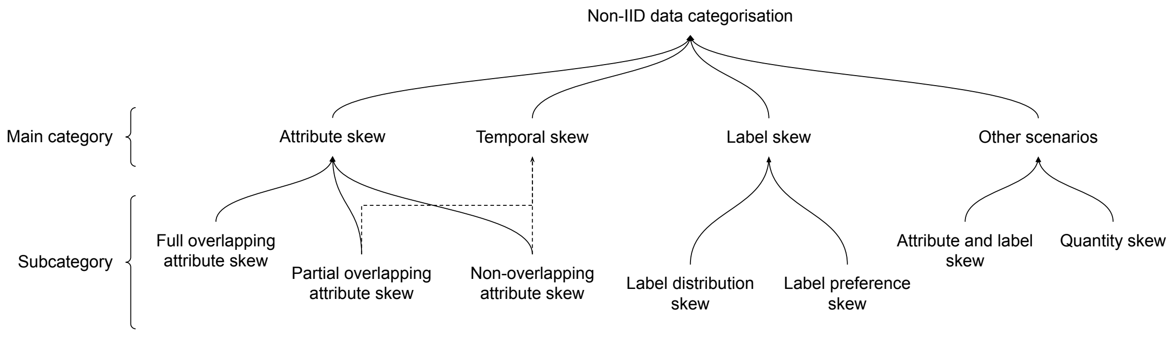


Figure 3.1: Non-IID data distributions, as categorised by the authors of [73].

We can find multiple analogies for the client distribution regime described in [33]. As the name suggests, 'attribute skew' corresponds to covariate shift. 'Label distribution skew' describes a single corner case of the prior probability shift, while 'label preference skew' refers to the issue of concept shift. The problem described by the 'temporal skew' category arises in time series data, and specifically concerns time shifts in data. It is therefore a mixture of the cases of 'non-overlapping attribute skew' for a single timestamp attribute and the 'partial overlapping' issue, as introduced by the authors of [73]. Based on the 'Other scenarios' example, we can assume that it discusses 'unbalancedness' category. Presented 'Attribute skew' and 'Label skew' relate to unstructured data comparison or mixing main categories.

The data categorisation presented above may help in better evaluating algorithms for processing video footage, as it is based on image and time-series data. Nevertheless, this categorisation is a particular case of the non-IID open problems for the image data and does apply strictly to such datasets only.

3.2 Uneven data distribution

Due to the widespread use of data distribution, the testing of algorithms with differently distributed datasets has attracted strong interest. In the literature, we find only a few available non-IID datasets that are already partitioned and ready to use for evaluation, called benchmarking datasets. Several methods of generating synthetic partitions have been presented, while current research is more focused on simulating the partitioning of existing datasets. In some methods, partitioning simulation interference with data attributes. Modifying the dataset is not recommended as this is similar to data generation, since either generating or perturbing data follows a similar predefined pattern of attribute modification.

3.2.1 Naturally partitioned datasets

Worldwide exist individual naturally collected non-IID datasets that have been published and are available. This situation is usually caused by data regulation and privacy concerns limitations; hence, this kind of dataset is difficult to obtain, as reported in [42] with reference to [28].

The authors of [48] published the first and only available non-IID dataset, consisting of images collected from differently located cameras additionally tagged with location and some metadata. Each image is labelled with a finite set of seven objects that can be found in them. In addition to the raw data, the authors provide two divisions of this dataset into client partitions, both of which simulate the category of 'unbalancedness', that is, an imbalance between clients where the data samples vary across partitions. The first considers the geographic locations of the nearby cameras, which are the sources of the images, and groups their images together on each partition. The second division separates each camera source as a single independent client with data. Both of these divisions naturally form 'prior probability shift' partitioning. The dataset is small and is dedicated to object detection in computer vision problems.

In another work [26], the authors analysed real-world examples of so-called 'skewed label partitions' in data. They studied the geographical distribution of mammal pictures on Flickr, and produced the *Flickr-Mammal* dataset: a non-IID partitioned dataset corresponding to geographic regions. In contrast to the dataset described above, this was artificially created according to the presence of each animal in a geographical region, and more effort was required to preprocess the data. Several interesting findings were discussed, such as the worldwide share of mammals' presence on each continent. For instance, the share of kangaroos and koalas in Oceania was 92%, suggesting that they are outliers or anomalies in data samples from other regions. A natural distribution of this type may be much more challenging to process and classify using decentralised algorithms. The authors also propose another data partitioning strategy in their work by controlling 'skewness' in specific data label occurrences on separate partitions. This prepared set was also an image-type dataset, similar to the previous one, and according to the example mentioned above, the data distribution relates to the 'prior probability shift' issue.

3.2.2 Generation of partitioned datasets

Generating means doing something from scratch, and similarly to the available datasets, a small number of works can be found in which synthetic data are created with uneven distributions. Multiple works have reported the evaluation of algorithms, usually FL algorithms, using real non-IID datasets being generated [44, 46, 41, 27]. At the same time, they are actually simulate "non-IID-ness", referring directly or indirectly to the paper in [52], which is one of the first examples of preparing non-IID partitioning. The authors of this work divided two datasets, MNIST [40] and CIFAR-10 [37], to form partitions with only a few classes of data sample, called pathologically partitioned non-IID data. From this work and the related references, we can observe the strong popularity of simulating different data distributions on certain datasets, such as MNIST [44, 52, 46, 41], CIFAR-10 [52, 46, 41, 27], and EMNIST[1] [44, 34].

The authors of [42], [12] and [44] demonstrated methods of generating synthetic non-IID datasets. In [12], the authors introduced a synthetic data generator that operated with two settings: task-dependent, and clustered around more than one centre. This was intended to cause current meta-learning methods to fail due to the additional heterogeneity. The method of generating a dataset proposed in [44] was based on a similar concept, and involved generating data partitions with more heterogeneity in the partitioned data. The authors also referred to the method of generating a non-IID set introduced in [62]. This basic generation method was created to verify distributed communication optimisation even before the FL term was defined in 2016 [36] in the way it is currently understood.

Since generation means creating something new, we can also classify methods that modify data attributes as data generators, since they change the characteristics of the original data. In a recent paper [42], one of the methods applied to achieve covariate shift in the form of feature distribution skew was the addition of noise to data attributes. The same approach to generating this kind of distribution was presented in [46]. The authors also introduced a method of generating their *FCUBE* dataset of synthetic three-dimensional data, which involved dividing the dataset into small cubes to form partitions with feature distribution skew.

3.2.3 Simulation of data distributions

Most works have focused on simulating uneven data distributions from the existing datasets for benchmarking, but have usually created division methods that are only applicable to their experiments. Each paper provides a more or less detailed description of the method used for partition generation, but operates on a fixed, predefined set of publicly available data. The most popular are image and textual datasets, including the following, which are referred to in multiple works: MNIST [42, 44, 52, 11, 46, 41], EMNIST [42, 12, 44, 34, 11, 68, 46, 41], Fashion-MNIST[2] [42, 11], CIFAR-10 [42, 52, 11, 46, 41, 28, 27], Shakespeare [12, 44, 52, 11], and Sentiment140 [12, 44, 11, 28]. Many authors have published implementations of their methods for dividing these datasets [42, 12, 44, 11, 68, 46, 41]; however, they are strongly connected with these fixed datasets, and the level of diversity in the implementation codes is

[1]The EMNIST dataset introduced in [15] extends the standard MNIST dataset with letters, making it more comprehensive.

[2]The Fashion-MNIST dataset introduced in [69] as authors wanted to replace the old-fashioned, too easy-to-process, and overused MNIST dataset.

high. Several works present only a description of the parameters used, which is difficult to reproduce [52, 34, 24, 27].

Studies of multiple distributions can be found in [42, 46, 41], where a systematic approach has been applied to target the open problems in FL. Nevertheless, most works focus on the covariate shift [12, 44, 52, 34, 24] and prior probability shift [12, 44, 52, 34, 11, 28, 27], and seldom the unbalancedness [52] of the data. The *skew* of features and labels is fairly common, and is an issue strongly related to the image processing tasks to which these works refer. The vast majority of works are aimed directly at highly distributed FL problems and divide datasets into large numbers of partitions, often forgetting that different techniques also suffer from non-IID data distributions.

Recently, several authors [42, 46, 41] have proposed methods of simulating multiple non-IID data distributions. In [42], three of the five defined open problems in data distribution were evaluated on both image and tabular (nominal) datasets with FL algorithms using different neural networks. The authors describe the proposed dataset simulation methods with different partitioning strategies. In their paper, label distribution skew is divided into the two subgroups of quantity-based and distribution-based label imbalance. The quantity-based method allows a fixed number of labels to be set for each partition. The highly extreme pathological case where each partition contains only samples with a single label is rejected by the authors but available to achieve as an extension of this approach. In the second approach, a Dirichlet distribution is used to divide data samples across partitions, which may be parameterised to allow the imbalance level to be changed in a flexible way. The same distribution function is used for quantity skew simulation. These two label distribution skew methods do not cover every prior probability shift scenario, and this is especially true for the first one, which is neither deterministic nor very customisable for simulating different cases of data distribution. The authors also divide feature distribution skew into several subgroups. The first assumes the addition of noise to the data, while the second refers to the *FCUBE* method of dataset generation, as described in Section 3.2.2 above. The last subgroup assumes that real-world feature imbalance follows a natural data distribution, provided by random and equal scattering. However, this assumption is generally incorrect, and may only be applied to a specific dataset where the data are split based on these independent features to achieve a non-IID distribution. When we randomise samples, we achieve a uniform distribution across partitions. Of course, this depends on the distribution function used with the order of the data samples, which were not mentioned. The authors do not consider the problems of concept drift and concept shift.

The authors of [41] considered data partitioning simulation for the same three main groups. They found that in practice, real datasets contained a mixture of non-IID categories, and hence performed experiments that combined feature distribution skew with label distribution skew partitioning, and both of these with quantity skew. Partitioning was based on choosing several classes per node and disturbing the balance between the numbers of data samples across the partitions. The authors neglected the aspect of feature skew by grouping data samples based on dataset-specific features, for instance, based on the writer for the MNIST dataset. Their explanation included a strong assumption that the MNIST dataset represented a typical example of feature distribution in view of its characteristics; however, this assumption can only be applied to concrete datasets.

In contrast, the authors of [46] considered the issue of concept shift but dismissed concept drift in similar way to previous approaches. The 'unbalancedness' method was briefly described, in which the sizes of the partitions, called 'shards', were set manually to determine the impact on processing; however, no results were presented. Prior probability shift partitioning was achieved using a combination of related parameters, such as the number of nodes and dataset size, but the explanation was unclear. For concept shift, the authors suggested dividing the main class of images from the CIFAR-10 dataset into multiple contexts describing the background. This method implies a different understanding of the data sample, in the sense of what is the recognised object on the image but requires manual or semi-automated work.

In [11], the authors introduced methods of simulating the unbalancedness, covariate shift and concept shift. The main problem here was that they confused the meaning of imbalance in terms of amount with imbalance in terms of class, by introducing the "random split" category. A second problem concerned the confusion of covariate shift and concept shift with prior probability shift for the first two types of the "split by label" category. Concept drift, defined as Type 2 of their "split by label" category, was a mix of covariate shift and concept shift. A pure concept shift is unavailable in their method, as two workers could not share the same sample with different classes. However, the division methods used were not described in detail. The two newly introduced categories were strictly related to the prior probability shift, as this is primarily associated with data division based on labels. A similar focus in this non-IID category can be found in [27], where this type of distribution was simulated using a differently parameterised Dirichlet distribution.

Several recent publications have documented different approaches to simulating non-IID data partitions. Most of these authors have prepared data for the evaluation of their algorithms [52, 34] or have introduced new benchmarking frameworks [12, 44], as discussed in the next section. Moreover, most of the works in this area consider image datasets and a performance evaluation of the different neural networks used in FL.

3.3 Benchmarking datasets and tools

In the literature, the term 'benchmarking' has multiple meanings. In most cases, it is assumed to refer to a complete tool that takes an algorithm as an input, simulates the execution environment, processes it and provides results that can be compared. The most difficult task is to implement such a tool to be generic and applicable to various algorithms. The opposite approach is to prepare a set of datasets with specific characteristics that exploit corner cases for algorithms, and this is one of the benchmarking categories mentioned in [43]. It is the most popular method of comparing algorithms using the same dataset and of providing results in terms of quality, memory and execution time. Nevertheless, this approach is becoming deprecated in distributed computing, where multiple data distribution possibilities, transfer load and privacy are important issues. A balanced and popular option is to create a framework combining both of these approaches. This requires a custom interface to be implemented, but it simultaneously provides the data and their different distributions. Due to the current popularity of FL algorithms written in the Python language, most of the recently developed tools focusing on this technique have been implemented in Python.

3.3.1 Dataset-based benchmarking frameworks

Very few publications refer to pure, ready-to-use datasets as benchmarks, and this is understandable due to the myriad possibilities in regard to data distribution. These works suggest a specific division of the data samples across partitions [48, 26] or directly describe the possibilities for customised distributions [12, 24, 46]. The first group concerns preparing datasets with specific characteristics that can emphasise the weaknesses of the evaluated method. The second group can be categorised into those similar to frameworks, where the dataset used in the evaluation and parameterisation of the distribution is an option that can be chosen. In contrast to other works, the authors of [24] focused on preparing a customisable non-IID dataset for benchmarking image classification algorithms, rather than creating a new framework.

Since FL has become the most popular method, and the issue of non-IID data distribution has started to be investigated, frameworks for this technique have followed the main language of the algorithms, which is Python. Hence, most evaluation frameworks are dedicated to FL algorithms and are written in Python [12, 44, 11, 46], although individual works can be found that refer to the evaluation of other distributed techniques, especially with a focus on dataset characteristics. Examples include Spark applications for benchmarking frameworks [9] and [66]. The PEEL framework proposed in [9] seems to be an interesting approach to testing; nevertheless, it is tightly related to the definition of a complex programming test suite, and does not consider the issue of data partitioning. The second of these works [66] introduced a benchmarking test suite with representative algorithms and datasets that allow for performance comparisons in general, including resource usage and transfer load rather than quality, and was dedicated to evaluating the Spark engine.

Most benchmarking frameworks in the literature with support for uneven data distribution offer the possibility of simulating a non-IID data distribution and evaluating algorithms on it, and collecting certain metrics as a result. The main issue with this approach is that these frameworks are limited to the small variety of datasets offered with the tool, and require individual, customised preprocessing. The datasets used in this paper were referred to in Section 3.2.3.

The authors of several works [42, 12, 44, 11, 68, 46, 28] provide a finite set of datasets that are available for their benchmarking tools. All of them are implemented in Python, and evaluate the performance of FL algorithms on these data with different distributions.

Many other tools are described in [43]; however, there is as yet no overall consensus on how to evaluate distributed algorithms properly, nor how to define metrics for the non-IID-ness of data.

3.3.2 Evaluation platforms

Non-FL distributed processing tools and algorithms have also been intensively developed across the world, but very few universal platforms are available, as described in [50]. In a similar way to FL benchmarking tools, they were created for the purpose of validating other DDM methods, and can be found in [45] or in our latest work [50] in this area, which is summarised in Section 4.3. We categorise such works as platforms, since they offer the user a more experimental approach and provide extensibility with custom algorithm implementations. The aim of using them is to experiment and compare, whereas the benchmarks described earlier focus on fixed test suites rather than practical usage. This kind of experimental approach is also offered by the TensorFlow Federated (TFF) platform [10] or PEEL framework; however, the first platform is restricted to FL-like algorithms and the second framework to Spark applications.

Unlike the frameworks described here and in previous sections, the authors of [45] focus more on decision trees than FL algorithms, although their approach is not limited to such methods. This promising system is based on the agent and artefact paradigm, and is highly configurable via an XML code called JaCa-DDM [45]. In this system, agents are responsible for orchestration and communication, while artefacts are wrappers for Weka [67] classes, as the JaCa-DDM system works around the Weka environment through the so-called 'agentification' of Weka for code reusability. The system itself meets many of the requirements for benchmarking: it collects processing times, traffic sizes during the training phase, and quality measures. It also strongly focuses on different execution (learning) strategies, which include communication methods and data transfer between agents and nodes. The data partitioning aspect is simplified to a manual data distribution configuration setup and a simple hold-out or cross-validation distribution. This choice was made because the authors noticed that the core problem in such agent-based DDM algorithms is not learning but collaboration. The limitations noted by the authors are that the simulation of distributed processing in which data is actually distributed in the experiments, while the agents are kept local on the same node. Data models that are strongly connected to Weka and MOA [8] and a lack of user-friendly configuration have also been reported as limitations. This approach focuses on the system orchestration, setup, and tear down environment, and neglects the simplicity of usage and the analysis of results. The aspect of data partitioning is bypassed, and is left to the user to perform manually.

Chapter 4

Partitioning methods and algorithm evaluation

The algorithm evaluation method presented in this chapter consists of several steps. The first step relates to the different data distribution methods and the simulation of these distributions. The second involves the proposed adaptation of a distributed algorithm as an example of how to achieve more insensitivity to an uneven data distribution. The third step relates to the platform used to prepare the evaluation environment and to execute the defined test suites. As a main contribution of this work, we propose an approach to testing with multiple data distribution test suites that opens up the possibility of examining the performance of the overall distributed algorithm. The data are not limited to a uniform distribution, and the results are not restricted to single metrics.

This chapter is divided into three sections corresponding to the abovementioned steps. The first is related to non-IID data partitioning methods, while the second presents an adapted hybrid version of a clustering algorithm that is robust to the data distribution, and the last is dedicated to the evaluation platform.

4.1 Data-distribution-based evaluation method

In the following subsections, we describe methods of preparing uneven data partitioning from a provided dataset for each non-IID main category. Our contribution is not limited to the proposed methods, and we also systematise their organisation.

4.1.1 Taxonomy for non-IID data partitioning

The taxonomy of non-IID data regimes introduced in [33] includes reasonably general categories, but requires standardisation and organisation in terms of the tree-graph structure. The basic version can be drawn as shown in Fig. 4.1. The taxonomy is strongly focused on FL issues when referring to general data partitioning issues.

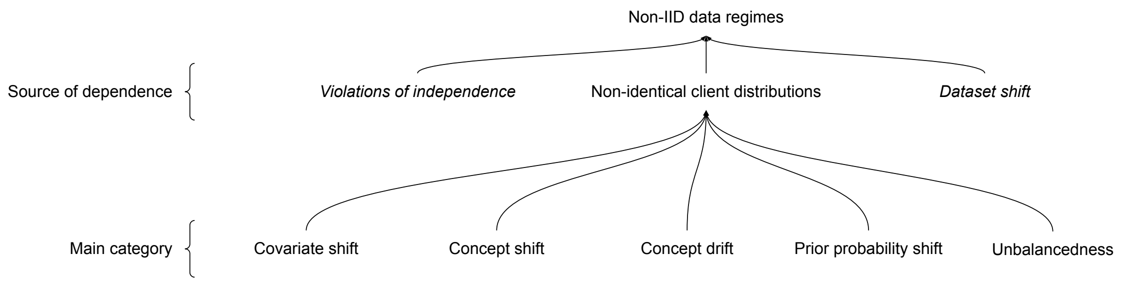


Figure 4.1: Illustrated taxonomy of non-IID data regimes.

The root of the taxonomy is divided into three 'source of dependence' paths, according to [33], which originate from the FL domain. 'Violations of independence' refer to the problem of missing access to several partitions during the training phase. In the general partitioning case, this maps to the situation of missing data (required or not) for the model preparation. 'Dataset shift' refers to the issue where training is carried out using data whose distribution characteristic is entirely different from target data where the built model suppose to be applied for. This also refers to missing data or the case where the wrong source partitions are selected for training. Those two issues are more strictly related to the FL technique, whereas our primary interest is in non-identical client distributions, as this refers strictly to the data distribution itself. Data can be divided across partitions horizontally or vertically, although the horizontal approach is more typical and hence is more closely analysed.

After a deeper exploration of the hierarchy, we noticed that this taxonomy was not detailed enough to determine whether testing an algorithm with a single distribution category is sufficient to say that the algorithm will work properly regardless of the data distribution. The various partitioning methods described earlier can be applied to simulate the same non-IID categories through the use of distinct techniques and modifying selected data properties. It is necessary to define additional levels to understand the hierarchy better and to avoid possible overlap between partitioning strategies for certain data distribution characteristics. The proposed taxonomy is illustrated in Fig. 4.2; a defined hierarchy shows data partitioning strategies associated with the main categories and their relationships to the properties of the data.

Each dataset consists of samples described by multiple properties. When analysing these data in terms of partitioning methods, we can extract three major properties. The basic one is related to the number of data samples, in which the number of data samples placed on the separate partitions impacts the processing performance or the initial statistical analysis. This plays a more significant role in horizontal partitioning than in the vertical approach, where it would be a partial data property in terms of partitioning. Vertical partitioning splits the entire dataset, and an analysis of the number of data samples would split it along a second dimension, which does not naturally fit the hierarchy presented here. The attribute relation can be applied to both horizontal and natural vertical partitioning. Different attributes are placed on different computational nodes when a dataset is partitioned vertically, whereas in horizontal partitioning, attribute

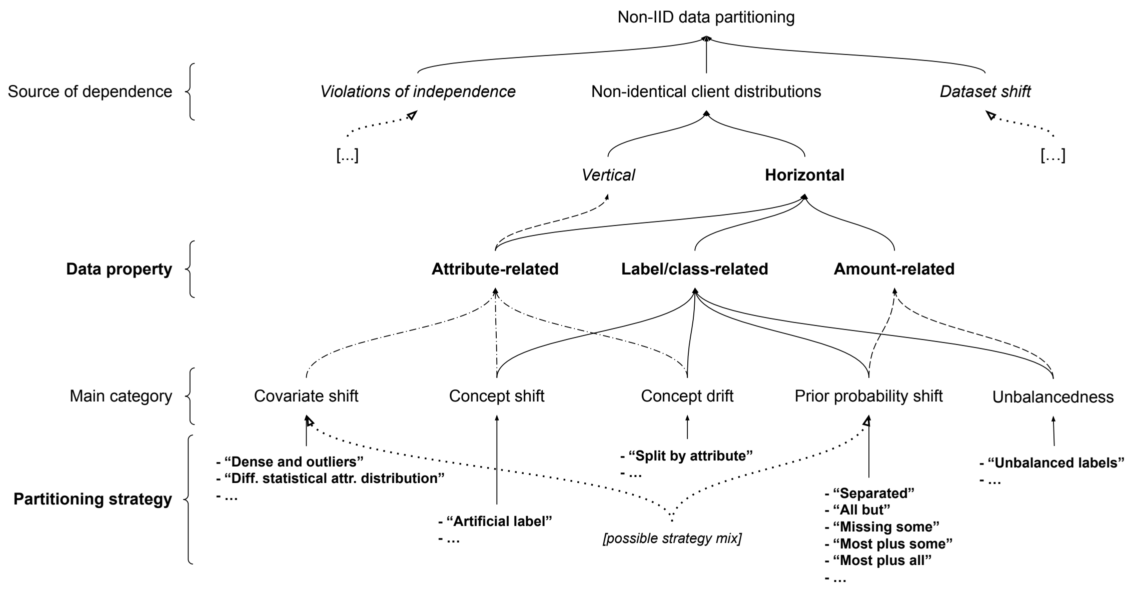


Figure 4.2: Extended non-IID data partitioning taxonomy, including new levels representing 'data property' and 'partitioning strategy'.

distribution plays an essential role and depends on the distribution characteristics, and has different impacts in the various types of algorithm used for processing. The last data property introduced in our taxonomy relates to the data sample class or label, which is treated as a special attribute. Unlike the attribute relation, it does not naturally fit with vertical partitioning, as the class attribute would not then be distributed over multiple nodes. The distribution of classes among partitions is crucial during the training phase, but plays a minor role in unsupervised methods.

The data properties described here form a natural bridge between the basic types of partitioning and the non-IID distribution categories introduced in [33]. They directly show the origin of the issue for those main categories. More detailed descriptions of the main categories are presented in later sections. None of these open issues relates to each category, and hence we cannot state that partitioning a dataset using a single data property will allow for a comprehensive evaluation of an algorithm. However, we can prepare dataset distribution methods that simulate all of these open issues. For the purposes of standardisation and better organisation, we introduce a specific new leaf level of the taxonomy, which represents partitioning strategies. A comprehensive algorithm evaluation should be executed for every strategy; however, scenarios of mixed partitioning issues are also present in the real world. For example, unbalancedness can be present in data partitioning classified for each main category. Sample strategies appearing as leaves represent separate methods of partitioning data, and these are explained in the following subsections.

The construction of this taxonomy offers a more detailed view of the possible data distributions. This knowledge allows us to create a comprehensive test suite covering multiple non-IID data distributions to evaluate algorithms under different circumstances. In addition, we can quickly see from this graph which main categories have been evaluated using specific data partitioning methods and which ones have not yet been evaluated. For example, the categories of covariate shift, concept shift, and concept drift relate to data attributes; however, they do not consider the quantity of data that might be required for testing the processing characteristics of the algorithm. On the other hand, a close relation to the data quantity in prior probability shift or unbalancedness does not concern data attributes values. We can distinguish a minimal test suite for algorithm evaluation based on these observations. Using non-IID data, we need to examine at least two of the five partitioning categories for that evaluation. Hence, to carry out a correct but minimal algorithm evaluation, we need to use at least one partitioning strategy that belongs to each data property in the taxonomy graph, since we consider different data distribution aspects in terms of data sample properties. An example set would be covariate shift and prior probability shift partitioning, which covers each data property at a higher level. An illustration of this coverage is given in Fig. 4.3. A different approximation of the evaluation might be a mixture of strategies used as a single test case to cover multiple data properties; however, it would exclude corner cases, which are vital for performing a comprehensive evaluation.

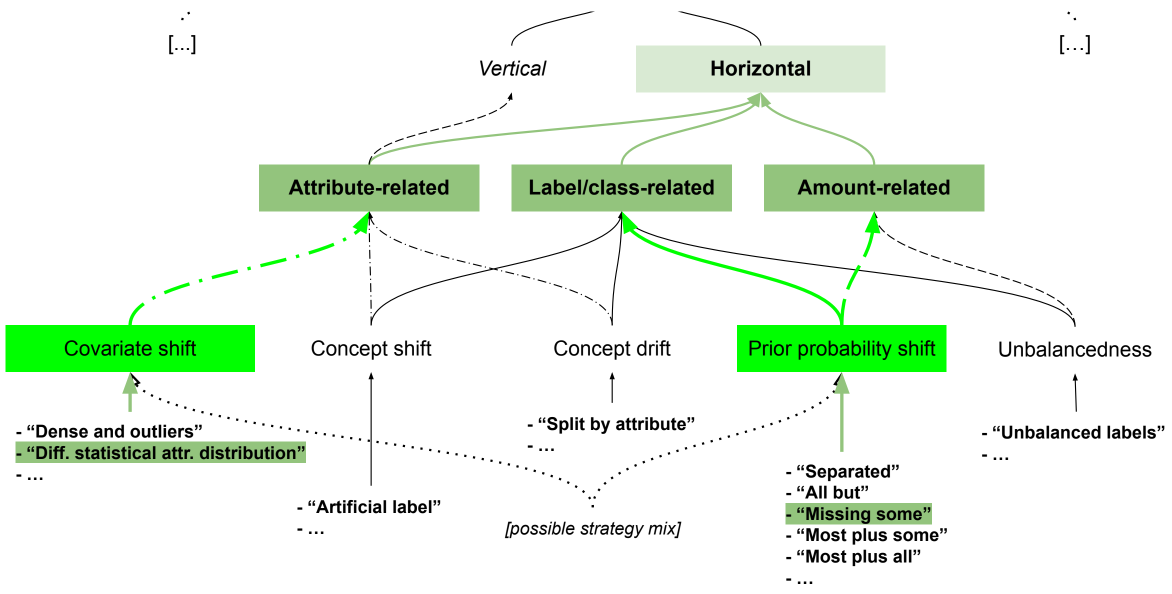


Figure 4.3: Clipped taxonomy graph with a sample minimal test suite for correct data property coverage (marked in green).

4.1.2 Non-IID data partitioning strategies

In this section, we describe approaches for partitioning a single dataset to obtain each of the main non-IID category data distributions. Methods of simulating different non-IID distributions were presented in a recent paper [51], and are described here in more detail. These strategies for creating different data distributions correspond to the open issues in regard to comprehensive algorithm evaluation, in order to enable verification of the results obtained with these distributions. Each method of splitting a single dataset into multiple partitions requires input in the form of a dataset with labelled samples, with a specified number of target partitions and strategy-dependent parameterisation.

Covariate shift (feature distribution skew)

Covariate shift occurs when data are distributed in such a way that the value distribution of a single attribute or set of attributes is strongly disjunctive between partitions. In other words, a combined histogram of data collected from each partition and grouped by origin shows a relative shift in densities between these groups. This is illustrated in Fig. 4.4 and described in more detail below. Although these groups consist of data samples with the same classes, the attribute distributions are slightly different.

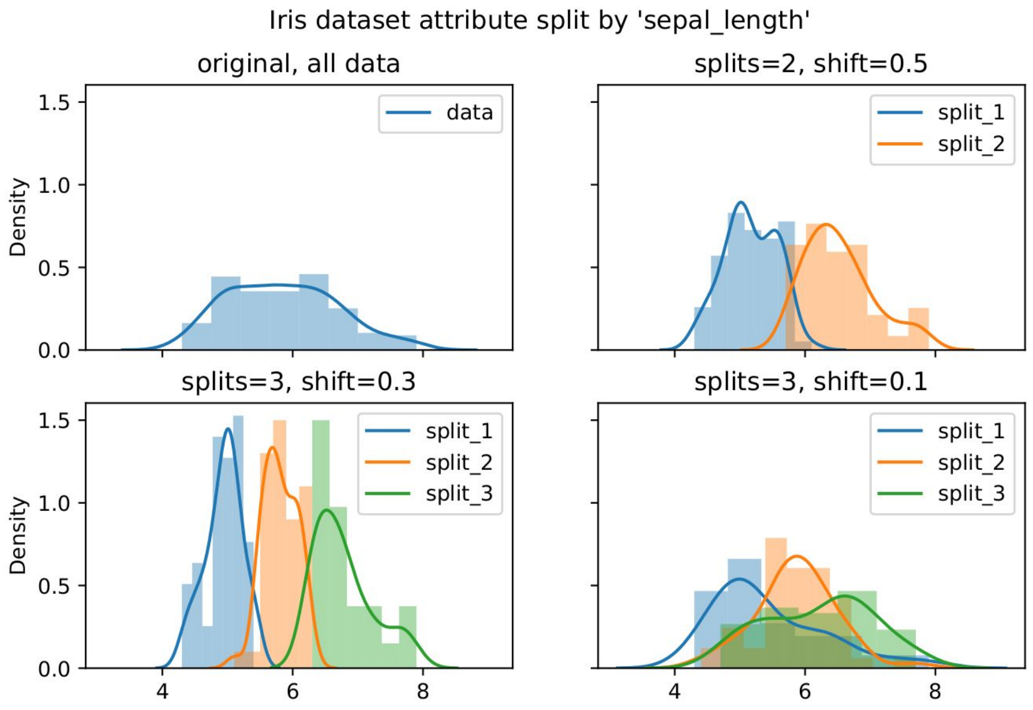


Figure 4.4: Examples of densities for the ’sepal_length’ attribute of the Iris dataset, in the original case and after applying the proposed splitting method.

The figure shows the original ’sepal_length’ attribute density for the entire set of Iris data. We can split the dataset based on this attribute in an attempt to achieve feature distribution skew. Then, depending on the number of target partitions and the desired skew size, we can obtain different attribute value densities for each partition. These densities vary, and the combined chart shows data sample attribute disconnectivity. Depending on the requirements, we could create more or fewer overlapping histograms to check that an algorithm handles each distribution in the same, correct way. Nevertheless, the possibility of splitting always depends on the characteristics of the data, and hence when there is low variation in the attributes, the number of possible disjointed splits is reduced.

Partitioning method It is easy to obtain nearly separate and evenly spaced data partitions based on a single attribute, by sorting the data according to the value of the attribute and dividing it into D equal parts. However, these parts have a data *shift* equal to $\frac{1}{D}$, which is the relative distance between the mean densities for these parts, as discussed below. Hence, to perform data splitting for different data shifts, we need to define an objective function for this problem. In our approach, we propose a function that consists of two user-defined parameters and unknown variables, defined as mean data values $M_n, n \in [1, splits], n \in N$ for each data split. The first parameter ($splits$) is the number of partitions, while the second ($shift$) is the distance between the data distribution means of the splits, which represents the target skew of the data. The objective function is determined by mathematical induction. We define the requirement for maintaining a certain ratio of the distances between the distribution centres of the splits as shown in Equation 4.1 for $splits = 2$, and in Equation 4.2 for $splits = 3$:

$$\frac{M_2 - min}{max - min} - \frac{M_1 - min}{max - min} = shift \equiv \frac{M_2 - M_1}{max - min} = shift \tag{4.1}$$

$$\begin{cases} \frac{M_2 - M_1}{max - min} = shift \\ \frac{M_3 - M_2}{max - min} = shift \end{cases} \tag{4.2}$$

Thus, the following equation is obtained:

$$\frac{M_i - M_{i-1}}{max - min} = shift \ , \ i \in [2, splits] \tag{4.3}$$

It follows from Equation 4.3 that the overall shift between first and last is a multiple of the *shift* parameter:

$$\frac{M_{splits} - M_1}{max - min} = (splits - 1) \cdot shift \tag{4.4}$$

We can rewrite the above expression as in Equation 4.5:

$$\sum_{n=2}^{splits} \frac{M_n - M_{n-1}}{max - min} = (splits - 1) \cdot shift \tag{4.5}$$

In this way, we obtain the final restrictions for the objective function as shown in Equation 4.6:

$$\begin{cases} \sum_{n=2}^{splits} \frac{M_n - M_{n-1}}{max - min} = (splits - 1) \cdot shift \\ \\ \frac{M_n - M_{n-1}}{max - min} = shift \quad , \quad n \in [2, splits] \end{cases} , \ splits > 1, \ shift \in (0, \frac{1}{splits}] \tag{4.6}$$

where *min* and *max* are the minimum and maximum attribute values.

Solving this problem with these constraints is not an easy task. Our goal is to minimise the difference between the defined target function and the selected ratio. To achieve this, we need to search the space of possible solutions, which grows with the amount of data. We therefore approach this as a typical search task that involves finding a solution that meets certain conditions. This can be done in many ways, including brute force, random methods, or evolutionary algorithms. In our implementation, we choose the simplest solution, which consists of two phases. We start by sorting the data based on the selected attribute value, to create an initial even split. We cannot assume that such a division satisfies a given condition or even that the distribution is close to complying with the requirements; however, it is a better starting point than a random order. If the attributes have a strict uniform distribution, after the initial split, we automatically obtain perfect objective function value matching parameterisation equal to $shift = \frac{1}{splits}$. Having sorted the samples, we then iteratively move a random sample from one split to another, rolling back the operation if the objective function deteriorates. The stop condition is defined in the standard way, as a low epsilon value or a maximum number of iterations. A search for a solution without rolling back the operation when the objective function value became worse caused continued deterioration, and this approach was therefore abandoned. Several strategies for moving the data samples were tested, including random splits and moving borderline data samples between partitions, but the simplest solution turned out to produce the best results. A random split refers to the initial state of the data samples. A moving borderline refers to a choice of samples that gives priority to those with edge partition values rather than random ones. Fig. 4.4 shows an example of the split obtained after applying the proposed method to a single attribute, and we can see how the distribution of the attribute values varies depending on the number of splits. Using a higher value for the $shift$ parameter, we obtain distributions that are widely separated with low overlap for each split. Each split represents a data distribution dedicated to an independent target partition. For a low value of the $shift$ parameter, the attribute distribution across the splits is more mixed, although the mean values are notably disjoint.

Splitting a dataset using a covariate shift for multiple attributes at the same time is possible, and requires that the function in Equation 4.6 is fulfilled for each attribute independently. However, the task then becomes more complex, and it may be impossible to achieve an acceptable function result for some datasets. Nevertheless, multiple experiments performed and described in Section 5 show that *shifting* data based on even one attribute affects the quality of the distributed algorithms.

Dense-and-outliers method The issue of covariate shift refers to different value distributions for the data attributes. It has a significant impact on training when the entire distribution is unknown on a single node, where unknown values disturb the model. At the clustering stage, it is crucial to know the neighbourhood of the samples, especially for density-based and hierarchical clustering types. We can imagine that the data are split into partitions with dense groups of samples and groups of data with the same label but treated as outliers or anomalies based on some similarity measure appropriate to the processed data.

By targeting the vulnerability in many density-based algorithms that cluster similar objects, we can design a partitioning method that exploits their nature. Using a similarity measure or the distance between points in multidimensional space, we can easily divide similar groups of data into dense and outlier partitions by finding the original cluster centre. We can then find the sample with the maximum offset from the centre, which acts as a base for partitioning. There are multiple ways of assigning samples to the dense and outlier groups; a simple approach assumes the percentage value of the maximum distance as a determinant, while a more sophisticated scheme tries to find the real density occurrence in the data samples. Both of these suggested approaches are illustrated in Algorithms 1 and 2. After the data have been divided into dense and outlier objects, we can distribute them among the specified number of partitions by gathering dense and outlier instances with different labels into a single partition using a uniform distribution. We can see an example of this process in Fig. 4.5, which shows the corresponding attribute density histograms. It is noticeable that this partitioning method gives a skewed distribution of the features, which is why it is located in the 'covariate shift' branch of the taxonomy.

Algorithm 1 Splitting data based on distance

```
Input $D \leftarrow$ data
Input $percent \leftarrow$ split percentage
Output $Dense, Outliers \leftarrow$ target
  $D_{C_i} \leftarrow$ data split by labels
  for all $D_{C_i}$ do
    $c_i \leftarrow$ find centre
    $d_{max} \leftarrow$ find max distance from centre
    $Dense_i \leftarrow \forall s,\ s \in D_{C_i},\ distance(c_i, s) < percent \cdot d_{max}$
    $Outliers_i \leftarrow \forall s,\ s \in D_{C_i},\ distance(c_i, s) \geq percent \cdot d_{max}$
  end for
```

Prior probability shift (label distribution skew)

Due to geographical or cultural differences, data samples from the same class collected by independent systems are often stored on independent nodes in different amounts. Based on this assumption and the examples described in Section 2.2.2, we can identify many possibilities for separating data distributions into independent data partitions. We can then distinguish the following partitioning strategies:

- '**Separated** [data]', where it is assumed that data samples with the same class are present only on one partition. This refers to the situation where data with a specified class stored at a node are only accessible on that node. The distribution learning aspect disappears for such a class, as all data samples are available locally; however, this may cause corner case issues when the data are processed by the algorithm;

Algorithm 2 Splitting data based on real densities

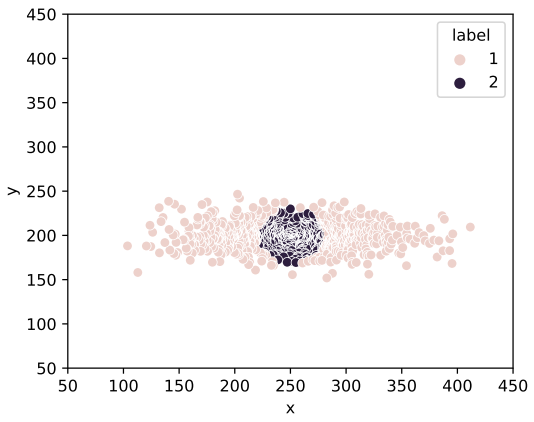
```
Input $D \leftarrow$ data
Input $percent \leftarrow$ density percentage
Output $Dense, Outliers \leftarrow$ target
  $D_{C_i} \leftarrow$ data split by labels
  for all $D_{C_i}$ do
      $c_i \leftarrow$ find centre
      $S_i \leftarrow$ sort $D_{C_i}$ data by distance to the $c_i$
      $d_{mean} \leftarrow$ mean of distances to $c_i$ in $Dense_i$
      for all $S_i$ do
          $updated\_D_{mean} \leftarrow d_{mean}$ change after adding next $s_i$ to $Dense_i$
          if $\frac{distance(s_i,c_i)}{d_{mean}} < percent$ then
              $Dense_i \leftarrow$ add $s_i$
              $Dense_i \leftarrow updated\_D_{mean}$
              continue
          else
              $Outliers_i \leftarrow$ rest of $S_i$
              break
          end if
      end for
  end for
```

- '**All** [of the data] **but** [some]', where specifically labelled data samples are missing from some partitions. This scenario assumes that data samples for all object classes except one or more for each partition are uniformly distributed when parameterised correctly. In this case, each partition does not have a complete overview of the class representatives that are globally present in the whole dataset;

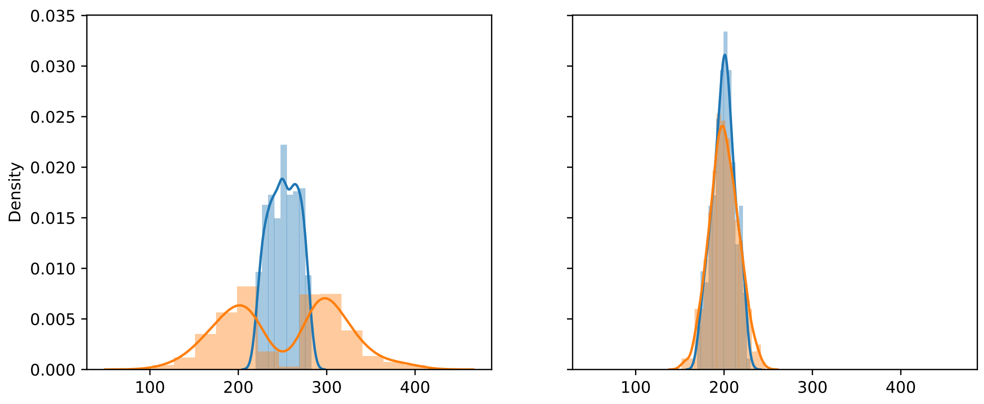
- '**Most** [of the data] **plus some**', where the data are partitioned according to their labels, and small subsets of them are then scattered among several independent partitions. Each partition is filled with a small portion of data samples from other classes in predefined numbers;

- '**Most** [of the data] **plus all**', which is the same as in the previous strategy except that each subset is also spread over each independent partition. This scenario simulates the case where each partition contains data for each label but with a different distribution;

- '**With anomalies**', where most data are partitioned evenly but there are anomalies due to certain labels in single quantities. These may be present only on several partitions or placed on each partition except the one with the majority of the anomaly class representatives.



(a)



(b)

Figure 4.5: (a) Example of partitioning a two-dimensional single Gaussian cluster using the 'dense-and-outliers' method; (b) density histograms for the X and Y features showing the split view.

All of these scenarios are based on scattering data samples across different partitions but with different and specific distributions in each case. The first three strategies assume main concentrations of data samples with certain classes and several classes missing within partitions. The last two strategies require representatives of all data classes to be kept together on every partition but in different amounts. Although it is possible to prepare a dedicated method for each label distribution skew, it all refers to the different amounts of particular data class representatives distributed across partitions depending on the selected case. Based on this and numerous experiments that were performed to obtain different distributions, we conclude that a single but parameterised method is sufficient to address each of the abovementioned strategies, rather than preparing separate, dedicated partitioning methods for each of them. An analysis of the results of our experiments showed that *skew* size is strongly related to the diverse percentage presence of the representatives on the partitions depending on the scenario. This is why we extracted these percentages as parameters of the partitioning method, as described in a later section.

Partitioning method The proposed method is divided into two stages, which are related to labels and data quantities. First, we separate each class or group of classes depending on the number of partitions, by assigning the classes in succession to the target partitions. The assignment is performed by iterating over the following partitions and distributing a given set of distinct classes, one at a time, with a cyclic iterator. The iterator is used because we want to avoid pairing classes on the same partition for different numbers of chosen splits. This approach produces many possible scenarios, and handles situations where there are significantly more distinct classes than target partitions. We then identify additional classes for the partitions and classes for empty partitions[1]. The additional classes on the partition correspond to the non-majority representatives that are present alongside the majority. The number of distinct class representatives is parameterised to achieve multiple different partitioning scenarios, as mentioned above. The method parameterisation assumes providing the number of additional classes to be placed on the partition or the number of missing classes on the partition determined negative parameter value. When the number of partitions is greater than the number of unique labels in the dataset, we need to choose how to fill the empty partitions, as there are insufficient unique labels for every partition. In our method, we have two options: to fill empty partitions evenly with samples with all labels, or to select a number of labels to be filled with data samples. The tricky part is avoiding adding the same label to every partition. We therefore use the concept of the cyclic iterator, as discussed above, by adding classes in order, which aligns the distribution of labels across the partitions. As a result of the first stage, we obtain tuples with three items: the main labels of the partition, additional labels, and supplementary labels used to fill empty partitions. Examples of the abovementioned partitioning methods are given in Fig. 4.6. The figure shows the final algorithm assignments of different labels, numbered from 0 to L, to each group of labels within D partitions.

In the second step, we use simple mathematical proportions based on the provided parameters and the amounts of data to be transferred from the main subsets. The mathematics is required because the minority and empty partitions must be populated with samples that are also assigned as majority ones for particular partitions. At the end of the process, the data samples are randomised with a uniform distribution for each label according to the defined dependencies, and then split between the partitions.

Both steps require several parameters related to the data class separation itself between partitions and the quantity of data samples to be distributed. The first pair is $additionalClasses$ and $emptyPartitionClasses$, which determine how many additional classes should exist in the partition despite the initial number of labels present in the partition, and how many labels to add to the empty partitions. The second pair is $emptyPercent$ and $additionalPercent$, which corresponds to the second stage and describes the maximum percentage of the data that should be used to fill empty partitions, and how much data should be used as additional padding for

[1] An empty partition is one without a dedicated majority of data samples for a specific label, which is possible when the number of unique labels is lower than the number of target partitions.

```
L - labels >= D - partitions

 EmptyAdd=0
```

```
Legend:
 1st [] - majority labels
 2nd [] - additional labels
 3rd [] - empty partition labels
```

```
L - labels < D - partitions
```

```
Separated:       d1=[0, 3] + [1, 2]       + []
 D=3             d2=[1, 4] + [3, 0]       + []
 L=5             d3=[2]    + [1, 3]       + []
 Add=0

All but:         d1=[0, 3] + [1, 2]       + []
 D=3             d2=[1, 4] + [3, 0]       + []
 L=5             d3=[2]    + [1, 3, 4]    + []
 Add=-1

Most plus some:  d1=[0, 3] + [1, 2]       + []
 D=3             d2=[1, 4] + [3, 0]       + []
 L=5             d3=[2]    + [1, 3]       + []
 Add=2

Most plus all:   d1=[0, 3] + [1, 2, 4]    + []
 D=3             d2=[1, 4] + [0, 2, 3]    + []
 L=5             d3=[2]    + [4, 1, 3, 0] + []
 Add=all
```

```
Separated:       d1=[0] + [] + []
 D=5             d2=[1] + [] + []
 L=3             d3=[2] + [] + []
 Add=0           d4=[]  + [] + [0, 1, 2]
 EmptyAdd=all    d5=[]  + [] + [0, 1, 2]

Separated:       d1=[0] + [] + []
 D=5             d2=[1] + [] + []
 L=3             d3=[2] + [] + []
 Add=0           d4=[]  + [] + [0, 1]
 EmptyAdd=2      d5=[]  + [] + [2, 0]

All but:         d1=[0] + [1] + []
 D=5             d2=[1] + [2] + []
 L=3             d3=[2] + [0] + []
 Add=-1          d4=[]  + [2] + [0, 1]
 EmptyAdd=2      d5=[]  + [1] + [2, 0]
```

Figure 4.6: Examples of different label partitioning strategies depending on the parameters *L* (number of unique labels), *D* (number of partitions), *Add* (additional number of labels), and *EmptyAdd* (additional number of labels for empty partitions).

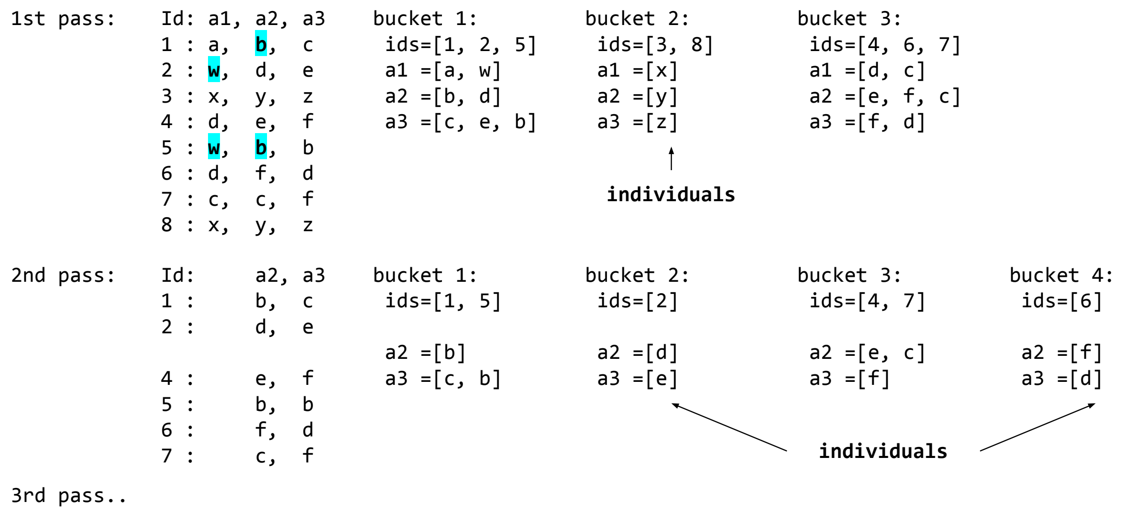
other partitions. To keep the majority in the initial subsets, the former should not be less than 50%, and the latter should be at most half the value of the former. The same concept is described in our recent paper [49] with reference to different strategies for preparing uneven data distributions.

The parameters have certain possible ranges; however, they can be set using small border values to obtain the abovementioned case with single representatives distributed over partitions representing anomalies on the data partition. There are many possibilities for parameterisation, which depend on the dataset used, the number of labels, and the amount of data. By looking at the final result of this partitioning process, we can clearly see that this process is closely related to both the data classes and the amount of data in terms of 'data property' from the extended taxonomy. In itself, it does not take attribute values into account; although this would be possible, it would involve mixing different main categories in a single partitioning strategy rather than a single aspect of the data distribution.

Concept drift (same label, different features)

The term 'concept drift' refers to a problem that arises in the training phase of distributed learning, in which a certain set of data objects from the same class on each independent node is represented by samples with disjoint attribute values. It can refer to a fully disjoint set of attribute values between nodes or simply a different value distribution for some attributes on different nodes.

Partitioning method To simulate this type of data distribution, the dataset with the selected label should be split to give the most separable data sample groups possible based on the attribute values. In other words, we split data to create sub-groups with possibly non-intersecting attribute values. A simple technique would be to apply a partitioning algorithm to a subset to find a specific number of groups; however, this may be a non-deterministic method, and may also require the specification of more parameters, which can change the results depending on the algorithm used. We took a different, deterministic approach, as summarised in Algorithm 3, which consists of four primary and several intermediate steps. First, we group data samples into so-called *buckets*, based on the intersection of attribute values. Then we find *buckets* with unique samples in terms of the single attribute values. Next, we exclude the least promising attribute for data sample division based on its entropy and repeat the preparation of *buckets* with a limited number of attributes. Removing a single attribute with lower entropy gives us more chances to find even more *buckets* due to the higher diversity of values; however, if we exclude any attribute during processing, we admit that data division with fully non-intersecting attributes is impossible. Nevertheless, because of their low diversity, these attributes may not play a crucial role in algorithm processing. Finally, we implement the partition creation step based on the found *buckets*.


```
1st pass:   Id: a1, a2, a3   bucket 1:         bucket 2:         bucket 3:
            1 : a,  b,  c     ids=[1, 2, 5]     ids=[3, 8]        ids=[4, 6, 7]
            2 : w,  d,  e     a1 =[a, w]        a1 =[x]           a1 =[d, c]
            3 : x,  y,  z     a2 =[b, d]        a2 =[y]           a2 =[e, f, c]
            4 : d,  e,  f     a3 =[c, e, b]     a3 =[z]           a3 =[f, d]
            5 : w,  b,  b
            6 : d,  f,  d
            7 : c,  c,  f                       individuals
            8 : x,  y,  z

2nd pass:   Id:     a2, a3   bucket 1:         bucket 2:         bucket 3:         bucket 4:
            1 :     b,  c     ids=[1, 5]        ids=[2]           ids=[4, 7]        ids=[6]
            2 :     d,  e
                              a2 =[b]           a2 =[d]           a2 =[e, c]        a2 =[f]
            4 :     e,  f     a3 =[c, b]        a3 =[e]           a3 =[f]           a3 =[d]
            5 :     b,  b
            6 :     f,  d
            7 :     c,  f                                          individuals

3rd pass..
```

Figure 4.7: Example showing the found *buckets* with *individuals* for the first two passes of the concept drift strategy. Bolded values with a cyan background are the attribute values used to group samples into the first bucket. The sample marked 'id 5' implicitly affects the association of the unrelated to each other samples with ids 1 and 2, as they have a transitive similarity: $1 \sim 5 \land 2 \sim 5 \implies 1 \sim 2$.

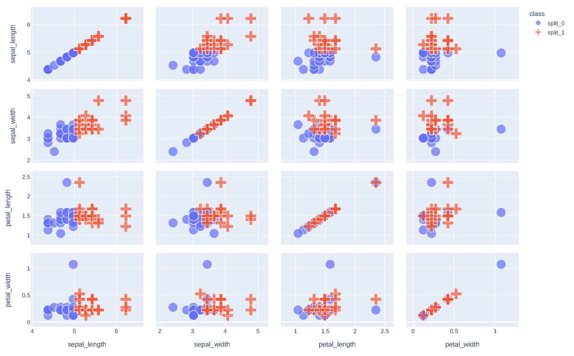
Although we assume here that *drifting* is performed for a single selected label, the same algorithm can be applied to other labels. Examples of the partitioning results for the Iris dataset and the two-dimensional numerical single ellipse Gaussian dataset are presented in Fig. 4.8. Using colours and shapes, it is easy to show how the data samples are divided separately from each other. Each group in the separated dataset represents the same object class, e.g. the

Algorithm 3 'concept drift' partitioning strategy.

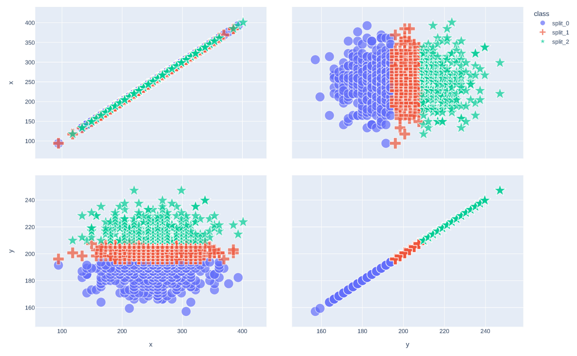
Input Dataset *Data*, target label *L*, and number of partition "drifts" *D*
Output *Data* partitioned into *D* partitions

```
1: Extract a data subset $Data$ with label $L$;
2: [optional] Discretise the numerical attributes into $R$ ranges for further processing. For categorical attributes, this step is not applied; however, it is required for numerical values, as real numerical values may all be different from each other, and the next step of the algorithm may fail due to the separation of too many similar samples;
3: Group data into buckets consisting of samples where at least one attribute intersects with another sample's corresponding attribute, to separate the individuals and groups of disjoint sets of attribute values. The individuals are samples or buckets of samples with unique values of all the attributes. A disjoint set refers to a group of samples where the values for each attribute in every sample are disjoint from the values of each attribute of every sample from other sets. Fig. 4.7 gives an example of this process, and shows the result and the second pass after processing Step 5.2;
4: [optional] For a dataset that contains only nominal attributes, find individuals. These are candidates for scattering anywhere as they have completely different feature values from the other samples, and can therefore be treated as data anomalies. Discretised numerical data can also be treated as individuals, but their target partitions are determined by the closest distance between them and the samples on the partitions;
5: Check whether it is possible to perform the selected 'drift' from the groups found by checking the conditions: $|buckets \setminus individuals| \geq D$ and $|bucket| > \frac{|Data|}{D} \cdot r; \forall bucket$; where $r$ is a ratio with a default value of 0.2, to avoid highly unbalanced partitioning:
    5.1: If the conditions are met, or all the attributes have been checked, partitioning begins as described in Step 6;
    5.2: Otherwise, remove individuals from the $Data$ subset, exclude the single $Data$ attribute with the lowest entropy value, and repeat the steps, starting with Step 3;
6: Partitioning is done by dividing the found buckets, sorted by the value of the last examined attribute, into $D$ partitions, where the number of samples is divided between them as evenly as possible in terms of quantity;
7: If any individuals are found, place them into partitions to fill or equalise the sample quantities between the partitions;
    7.1: [optional] For numerical data, the target partitions for individuals are those closest to the centre in terms of the Euclidean distance (note that without the discretisation performed in Step 2, there is usually a large number of individuals).
```

same genre of flower or cluster of points; however, these are divided as much as possible after examination of the attribute distributions. As a result, we obtain well-separated subclasses.



(a)



(b)

Figure 4.8: Scatter matrix plots for single label data partitioned using the concept drift strategy: (a) the Iris dataset, based on the 'Iris-setosa' attribute; (b) a two-dimensional single cluster dataset.

This approach may fail if the dataset cannot be split in this way, in which case the processing method is reduced to splitting the data sorted by the most diverse value of a single attribute. In this scenario, running a clustering algorithm with a fixed number of target groups on the data with the same class would be beneficial, and would simulate concept drift. However, this would also require parameterisation, and may not be deterministic. For images or textual datasets, the proposed approach should also consider either a prior aggregation of attributes or a reduction in the number of dimensions. Without this preprocessing step, the algorithm would try to carry out an analysis at too low a level of detail, such as a single pixel value, which is irrelevant in the overall scope.

Concept shift (same features, different label)

This category of non-IID distribution refers to the situation where data are understood differently or are misclassified in the training phase. When classifying non-textual data, it is difficult to avoid errors in processing, and it is almost impossible to evaluate the correct solution, as training is based on previously known labels. When the features of data objects are similar but labelled differently, the only way to get a correct result is to use fuzzy classification, where the classified sample can have multiple classes assigned to it. However, the clustering task is not affected by this problem, as it is an unsupervised learning method that does not involve an analysis of the training data classes.

Partitioning method In view of the considerations described above, the proposed method of data partitioning consists of changing the label to a new, unique one for a certain number of data samples. Our implementation of this partitioning strategy allows us to change the label of the training samples for a subset of data D with a selected class c. This training subset of size $s \cdot \frac{1}{N}$ is then separated and the labels are changed, where s is the number of parts of the data subset on which we perform an independent label change, and N is the number of partitions. Hence, we obtain the formula in Equation 4.7 for the number of samples with artificial classes:

$$|artificialSamples| = \frac{s}{N} \cdot |D_c| \tag{4.7}$$

Fig. 4.9 gives an example of this type of partitioning. The dataset in this case consists of data samples with two different labels, which are to be divided into three partitions, and two more classes are to be created using concept shift. As a result, we obtain four classes distributed on three partitions, where two classes are artificially created from the chosen one.

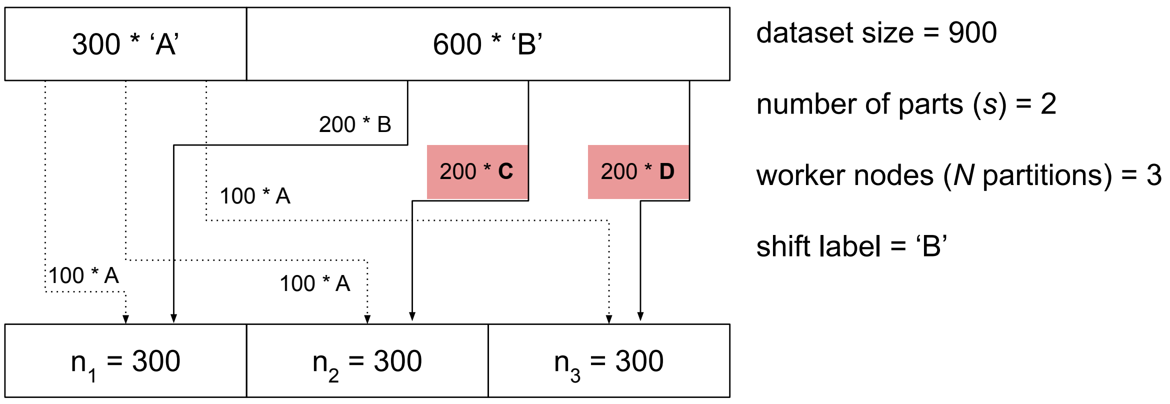


Figure 4.9: Partitioning strategy applied to data samples labelled 'B'.

There are other possibilities for creating a concept shift in data, such as generating new data samples based on the attribute distribution or adopting the concept drift method with a label change. However, the first method assumes attribute modification, which affects the provided dataset, while the second slightly simplifies the target problem but is also a good candidate for a mixed strategy. The overall purpose of using this method for validation is to examine the extent to which this divergent understanding of object class affects the final results. This can be useful when the evaluated algorithm produces a fuzzy result or returns artefacts describing samples rather than a simple classification label.

Unbalancedness (quantity skew)

Another open issue in the field of data partitioning is an unbalanced quantity of samples between the partitions processed by the algorithm, either in the training phase or without training. Splitting a given dataset into partitions with different numbers of samples can be done in multiple ways. The most trivial is using random data partitioning along with the numerical weights describing the target percentage of samples in each partition. However, it would be difficult to maintain the ratio between the sizes of these partitions with more data samples and other partitions for various numbers of target partitions. A simplified diagram illustrating the parameterisation used in this approach to obtain a ratio of $\frac{1}{4}$ is shown in Fig. 4.10.

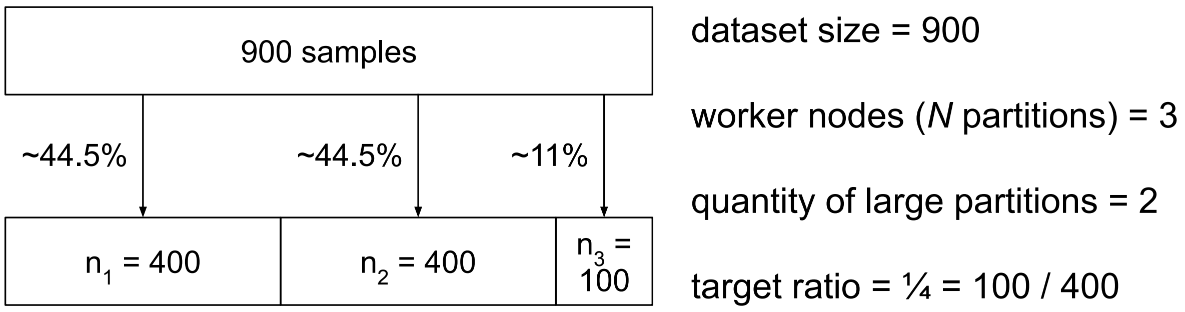

900 samples
~44.5%
~44.5%
~11%
n1 = 400
n2 = 400
n3 = 100
dataset size = 900
worker nodes (N partitions) = 3
quantity of large partitions = 2
target ratio = ¼ = 100 / 400

Figure 4.10: Trivial approach to finding the partition size division.

The problem of differences in data quantity starts when the number of samples between partitions differs significantly, as higher numbers of samples usually increase the processing time, while small amounts of data may disrupt the global model built by the algorithm. This is why the issue of unbalancedness arises when at least one computational node contains significantly less or more data than the others, meaning that not all of the partition sizes need to differ. Moreover, the difference between low and high data quantities depends on the size of the problem. This difference should therefore be represented in relative terms, for more flexibility when simulating such partitioning, rather than using providing fixed numbers.

Partitioning method Our aim was to allow the user to choose this ratio, along with the number of nodes that should contain more data samples, instead of defining the required amounts of data for each partition. We wanted to control the ratio between the high and low amounts of data on partitions to make it easy to set up in a controlled way. In order to make this possible, a simple system of two equations needs to be solved, in which we have two unknown variables and four known parameters. Equation 4.8 describes the expressions that need to be solved in order to find the unknown values L and S, where L is the large amount of data and S is the small amount of data; the known parameters are C and W, where C is the dataset size and W is the number of target partitions; and the user-defined parameters are the node threshold T, which is the number of large partitions, and the ratio U, which is the target imbalance.

$$\begin{cases} T \cdot L + (W - T) \cdot S = C \\ \frac{S}{L} = U \end{cases} \implies \begin{cases} L = \frac{C}{U \cdot (W-T)+T} \\ S = \frac{C - T \cdot L}{W - T} \end{cases} \tag{4.8}$$

As shown in the previous example in Fig. 4.10, we can obtain these numbers by using parameter values of 0.25 for the unbalancedness ratio and a threshold of two nodes. In our implementation of this method, it is also possible to select a proportional imbalance to keep the same ratio for each class as a result of data partitioning. However, this behaviour is usually undesirable, as it tends towards an IID data distribution. As a result of this partitioning strategy, the data samples are scattered among the partitions based on the calculated partition sizes, L and S. Samples are chosen from the original dataset using a uniformly random function.

It should also be noted that when the imbalance factor is high, we do not consider unbalancedness alone, but in conjunction with the category of prior probability shift. In this case, it is related to the 'label/class-related' data property defined in our taxonomy, because small partitions might have missing representatives of some classes. This is another indication that we are considering a taxonomy rather than a hierarchy.

4.2 Adapted and robust clustering algorithm

We present a novel hybrid concept that utilises two different types of clustering algorithms for single distribution processing: k-means is used as a base partitioning algorithm, and a density-based OPTICS algorithm is used to build global clustering models. The input parameters for OPTICS are determined automatically, based on our local clustering models, and no user input is required. The use of a density-based approach provides several advantages: we eliminate the influence of the bad start problem, so that even a random initialisation at the local stage does not decrease the quality of the results. In addition, it can also improve the global phase by calculating the real number of clusters for the entire distributed dataset.

In this section, we present an adapted version of a hybrid algorithm that handles non-IID data distribution correctly, and briefly describe the reference method. An exhaustive examination of the proposed hybrid algorithm and a performance comparison against the non-distributed execution of the core algorithms is available in the full paper in [49]. However, we present an additional algorithm evaluation based on the proposed partitioning strategies in this work.

4.2.1 Distributed k-means

In this section, we introduce the concept underlying the distributed k-means (DK-means) algorithm with a central coordinator, as proposed by [31]. This straightforward implementation of the DK-means algorithm does not consider the fact that data may be unequally dispersed, meaning that the densities of clusters on different nodes vary. This may cause the algorithm to fail to correctly identify clusters, as the non-trivial issue of finding initial centroids arises at the global processing stage.

The proposed approach consists of four stages of processing. In the first stage, all local datasets are clustered using the k-means algorithm (Process 1 in Fig. 4.11). The resulting centroids are treated as local clustering models, as this offers a reasonable compromise between the level of complexity and the precision of the calculation. Next, all local centroids are transferred to the central node (Process 2 in Fig. 4.11) and combined into a single set. This set is then clustered using the k-means algorithm to find global centroids (Process 3 in Fig. 4.11). Following this, the resulting centroids are resent to the local nodes (Process 4 in Fig. 4.11) and are treated as the globally correct, final centroids, based on all local datasets. In the final phase, local objects are assigned to the closest global centres (Process 5 in Fig. 4.11) based on their Euclidean distances. The time complexity of this approach is similar to local processing; however, in this case, the processing is divided into small distributed chunks, as described later. The overall concept of the algorithm is illustrated in Fig. 4.11.

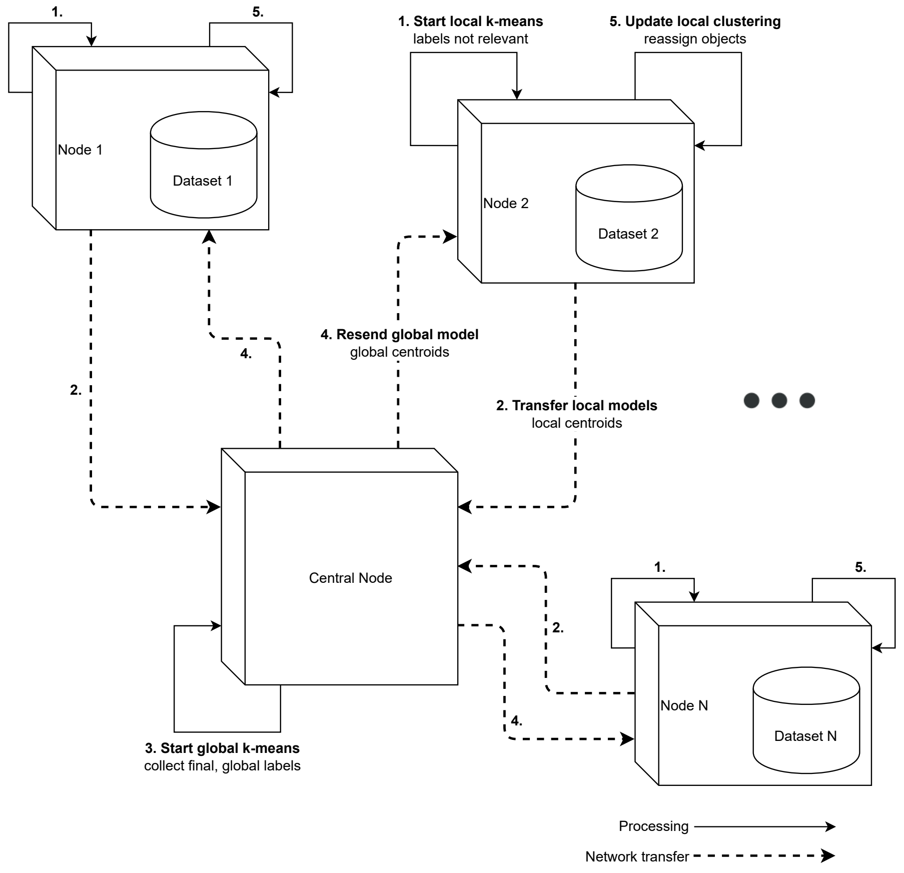


Figure 4.11: Illustration of the concept underlying the DK-Means algorithm.

This algorithm seems to provide suitable results for many datasets. However, the authors required that the user should specify the number of clusters to find on each node separately; this cannot be applied under real conditions, as it makes the algorithm unnecessarily complicated. Moreover, the DK-means approach does not appear to handle uneven data dispersion, an aspect which was neglected by the authors. The global phase can therefore be greatly improved by applying a density-based method. A standard k-means algorithm requires the user to provide a fixed number of clusters to be found. Assuming that we have a small number of local nodes, we then have a small number of models in the central node and the termination condition will be met within a few iterations. This result may not necessarily be correct, since initialisation at the central node in the k-means algorithm causes a problem that cannot be fixed by sampling a small number of objects. On the other hand, when the number of nodes is large and not all nodes contain a representative object of each group, searching for the same fixed number of groups is not a suitable approach. As a remedy, a density-based examination of the data may be helpful, as this could eliminate the need to define an exact number of groups.

4.2.2 OPTICS-Distributed k-means

The approach in our hybrid algorithm is consistent with the general concept underlying distributed clustering algorithms such as DK-means, although it retrieves statistical information from local models and makes use of it in the global phase with the help of the OPTICS [4] algorithm. This allows proper handling of the varying local density and automatic adjustment of the number of clusters. The general concept is illustrated in Fig. 4.12. For the purposes of this paper, we refer to this as OPTICS DK-means (Opt-DKM). The following sections provide a detailed description of the processing steps involved in our algorithm.

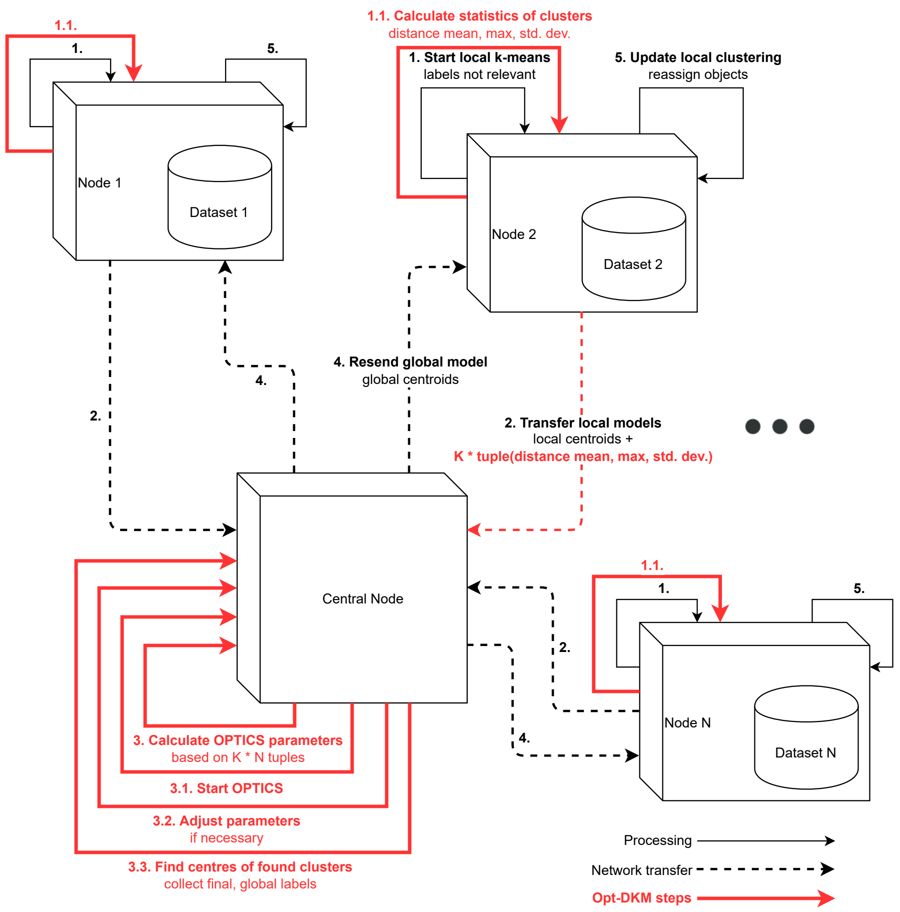


Figure 4.12: Illustration of the process used in the Opt-DKM algorithm; steps characteristic of Opt-DKM are marked with bold red lines.

Local clustering statistics model

In the reference method, a model for local clustering consists of nothing besides k centroids. However, in our approach, we enrich this with additional data. The local model is a tuple $(c, v_1, \ldots, v_n)$, where c are the attributes of the centroid and v_1, v_n represent statistics describing each cluster, as described in Process 1.1 in Fig. 4.12, including:

- The average distance between objects and their centroids;
- The maximum distance to the centroid;
- The standard deviation of the distance between objects and the centroid.

Assigning specific cluster IDs is not necessary, since the final labels will appear in the next (global) phase. Calculation of this model is straightforward, and carries much useful information. The formulas used are presented in Equations 4.9 and 4.10:

$$\vec{\sigma} = \sqrt{\sum_{i=1}^{N} \frac{(\vec{X}_i - \vec{M}_i)^2}{N}} = \sqrt{\frac{\vec{SS} - 2\vec{M} \times \vec{S} + N \times \vec{M}^2}{N}} \tag{4.9}$$

$$\vec{S} = \sum_{i=1}^{N} \vec{X}_i, \quad \vec{SS} = \sum_{i=1}^{N} \vec{X}_i^2, \quad \vec{M} = \frac{\vec{S}}{N} \tag{4.10}$$

where σ is the standard deviation; N is the number of objects in a cluster; M is the average distance to the centroid; X_i is the distance between the object and centroid; $\vec{S}$ is the sum of the distances; and $\vec{SS}$ is the sum of the squared distances.

Preparation of the global model

The essential innovation of our distributed clustering scheme based on Opt-DKM is building a global model for distributed clustering. The local models collected from the central nodes are treated as an input dataset for density-based clustering. The OPTICS algorithm is applied (Process 3.1 in Fig. 4.12) to this data to give clusters with a dense concentration of local models. As a result, the centres of the new clusters are calculated as the mean of the cluster members' attributes, forming new global centroids. However, in order to run the OPTICS algorithm, the user needs to provide the initial OPTICS parameters, such as ε and ε', as described in [4]. A major drawback is that these values do not correspond to the raw data at local nodes but to their model-based representations, meaning that the values of these parameters should be higher.

Our method determines the OPTICS parameters for input based on the collection of local statistical models (Process 3 in Fig. 4.12). The main motivation for this approach is to eliminate the need for the user to provide the parameters and to try to determine these based on the available data. This is important because the correct parameters for the density-based algorithm require at least an approximate knowledge of the characteristics of the data. The input parameters needed by the OPTICS algorithm are calculated based on simple formulas, as shown in Equation 4.11:

$$minPts = 1, \quad \varepsilon = \overline{d(c_i, o_i)_{max}}, \quad \varepsilon' = \varepsilon - \overline{\sigma_i} \tag{4.11}$$

where *minPts* is the minimum number of points in the neighborhood; $\overline{d(c_i, o_i)_{max}}$ is the mean of the maximum distances provided in the local models; σ_i is the mean of the standard deviations provided in the local models; and i is the next local model. The value of ε is calculated based on the maximum distances to allow for expansion of the density-based algorithm to adjacent centres and to gather them into groups. This computed value enables the connection of adjacent clusters, while remote ones are unaffected. The value of ε can also be considered as an initial k-means radius for the centroid. If two models truly represent separated groups, the maximum value will be less than the distance between them, and as a result, they will not be clustered together. On the other hand, if the models represent data belonging to independent nodes, then both of them will be reachable in this neighbourhood. The OPTICS algorithm also requires the parameter *minPts*, which describes the minimum number of objects needed in a group to create a new cluster. However, since we are operating on models rather than single data objects, we can set this parameter to a fixed value of one, since each object clustered in the global phase represents a potentially larger group of local objects. There is also no noise in global OPTICS processing. The parameter ε' was adjusted in the course of the experiments for different datasets, with the aim of separating noise from the dense groups. This prevents the OPTICS algorithm from merging it with regular groups in the global phase.

Potential border drawbacks

During the experiments, two potential drawbacks were identified that were related to the data, and corrections were developed to deal with them. A density-based approach allows us to find a real (in the sense of density) number of clusters, but when a node contains a single representative of a group and all groups have the same density, then it is impossible to merge the local clustering models in the global phase. This represents the 'separated' partitioning strategy from the 'prior probability shift' category, which is a strict corner case. Although this situation is rather unrealistic, our solution can accept additional parameterisation in an attempt to fix this potential issue. When a suitable parameter (*exactKGroups*) is set, we merge pairs of the closest centroids after the global clustering phase, until the expected number of sought groups is obtained. A second issue relates primarily to real datasets and the use of an unsuitable similarity measure. When a Euclidean measure is used for data across the domain, the calculated OPTICS parameters are often too high, and only a single large group is found. The solution for this is an iterative manipulation of the value of ε' until more groups are found, since clustering should never produce only one group. Since we do not need to rerun the clustering, if we set the *noOneGroup* or *minKGroups* parameters, a new clustering assignment is carried out and the previous value of ε' is decreased as shown in Equation 4.12 (Process 3.2 in Fig. 4.12). The first parameter is equivalent to $minKGroups = 1$, which targets the minimum number of groups to be found in the next rounds of the OPTICS clustering assignment.

$$\varepsilon' = \varepsilon' \cdot \frac{1}{2} \cdot \left(1 + \frac{mean}{max}\right) \tag{4.12}$$

Parameterisation

Both of these distributed methods require parameterisation at the start of processing. In the classical DK-means approach, the user needs to provide a global *k* parameter for the last global processing stage and the same parameter with a potentially different value for each local node clustering, which is not practical. The use of a density-based approach in the global stage avoids this requirement, and allows the user to set only the global number of groups, which will possibly be adjusted by a density feature of our approach. Nevertheless, in both approaches, setting a value of *k* that is too low may affect the process of finding centroids outside of data clusters and lead to the merging of groups of different objects. On the other hand, setting a value that is too high lengthens the processing time and may disrupt the calculation of the OPTICS parameters based on cluster statistics due to the small sizes of the clusters. A higher *k* parameter for local nodes might also provide more data for the global k-means stage in the reference method, which can be beneficial. In both the DK-means and Opt-DKM methods, the results depend on the number of nodes, the data distribution, initialisation of the algorithms, etc. Hence, an overestimate of the value of *k* is always better than an underestimate, as clustering algorithms by their nature combine data rather than divide. In general, the DK-means method finds the number of groups specified by the user, while the Opt-DKM tries to adjust the results to the nature of real data and is therefore more flexible in the case of inaccuracies input by the user.

4.3 Comprehensive evaluation platform

There is an ongoing discussion concerning how to verify the correctness of an algorithm in a standard way. As mentioned in previous sections, there is a need to test such algorithms against other methods under similar conditions. The first option is to develop a rigid method with preset regulations for testing and constraints that must be maintained by the creators of the algorithm during evaluation. However, it is not possible to validate such an evaluation. In addition, the provision of an algorithm verification or benchmarking platform raises several challenges concerning both flexibility and usage simplicity for end-users. Nevertheless, it is the only possible method of standardisation in which the results would be comparable.

A major drawback of many studies is a lack of information about the similarity measures used in clustering and with certain classification algorithms, which makes it almost impossible to reproduce results on a specified dataset. Another fundamental limitation is that they do not address the problem of data distribution during evaluation of the algorithm. In this section, we describe our proposed system for the evaluation of DDM algorithms, called the DDM-PS-Eval platform. The description consists of the general architectural structure and the main components of the entire platform, which are discussed in more detail in a recent paper [50].

4.3.1 Architecture

A system for evaluating different algorithms under multiple environment configurations, datasets and execution parameters should be both simple to use and provide a wide variety of functionalities. The main functionalities are as follows:

- Loading of algorithms and similarity measures;
- Loading of datasets and partitioning strategies;
- Environment configuration and parameterised executions;
- Collection of results and multiple execution statistics.

To make the platform useful, certain additional non-functional requirements are also essential, as follows:

- A user-friendly interface and flexible configuration;
- Simplicity of use, environment setup and comparison of results.

The environment should simulate a real distributed system in which data are accessible only locally for each computational node. To achieve this state, we can approach the problem in two ways. Firstly, we can simulate this processing distribution using sequential execution, with possible parallelisation. The second approach is to connect to independent machines or to create independent containers with appropriate resources. Both options have certain advantages. In the first case, we have all of the resources in one place, and there is no introduction of networking problems; however, we would still need to simulate the environment, which would make it hard to encapsulate the executions in terms of the available resources. This approach also raises scalability issues. The second solution gives us more configuration possibilities in terms of available resources to connect to the system. We have a greater opportunity to limit access and collect networking statistics. Nevertheless, such encapsulation suffers from orchestration problems and requires synchronisation of the provided resources that are evaluated, such as an algorithm codebase or distance function implementation. Although it appears that a local solution might be a better choice, it has less potential for future extensibility. We decided to use the second option, due to the configuration possibilities, the creation of a clean solution, and

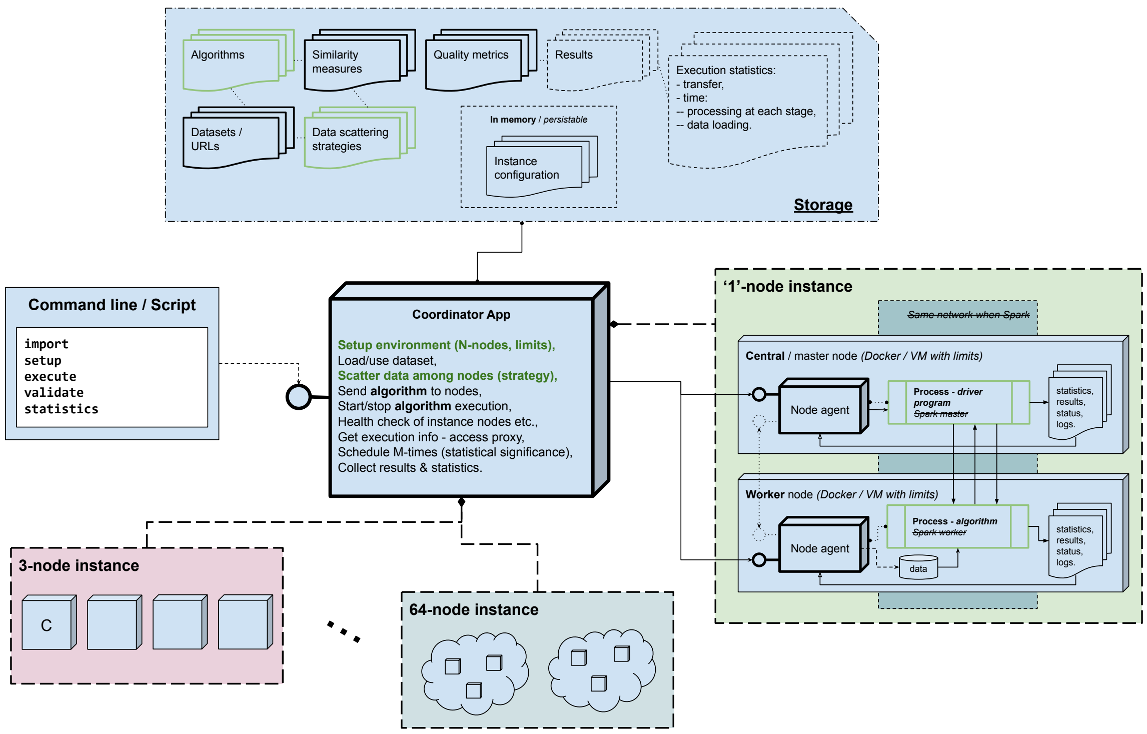

Algorithms
Similarity measures
Quality metrics
Results
Execution statistics:
- transfer,
- time:
-- processing at each stage,
-- data loading.
Datasets / URLs
Data scattering strategies
In memory / persistable
Instance configuration
Storage
Command line / Script
import
setup
execute
validate
statistics
Coordinator App
Setup environment (N-nodes, limits),
Load/use dataset,
Scatter data among nodes (strategy),
Send algorithm to nodes,
Start/stop algorithm execution,
Health check of instance nodes etc.,
Get execution info - access proxy,
Schedule M-times (statistical significance),
Collect results & statistics.
'1'-node instance
~~Same network when Spark~~
Central / master node (Docker / VM with limits)
Node agent
Process - driver program
~~Spark master~~
statistics, results, status, logs.
Worker node (Docker / VM with limits)
Node agent
Process - algorithm
~~Spark worker~~
data
statistics, results, status, logs.
3-node instance
C
64-node instance

Figure 4.13: High-level design of the overall system architecture.

the chance to create a tool that is coherent with current development principles. To illustrate the overall architectural concept of the system, the high-level design is depicted in Fig. 4.13.

Comparing systems from the same class of benchmarking allows us to see the advantages of the new platform, as it is hard to read from the technical components that non-functional requirements are met. The frameworks or systems mentioned here differ significantly from each other in terms of architecture and functions, and a complete comparison is not possible due to these differences; however, some features and facts about these functionally similar systems are summarised in Table 4.1. The crucial components required in order to improve the verification process of an algorithm are described in the following sections. A brief description of the possible platform environment configuration is attached in Section 7.1.2.

As reported in [45], the core issue with regard to DDM algorithms is collaboration, and this applies both to the execution of the algorithm and communication within the distributed systems. In our solution, the key component allowing for communication is an application installed on each computational node. This application handles incoming traffic, provides partial results and manages both the execution of the algorithms and the node itself, according to the agent paradigm. More technical details related to communication in the platform can be found in the attachments in Section 7.1.1.

Table 4.1: Comparison of convergent elements for the reported benchmarking systems with the proposed system.

	PEEL (framework)	**JaCa-DDM**	**DDM-PS-Eval**
Main purpose of the system	Reproducible experiments	Evaluation of learning strategies	Algorithm evaluation and comparison of results
Applicable algorithms	Any (custom); Hadoop/Spark-based impl.	Classification; WEKA impl.	Classification, clustering; implemented using wrappers
Environment setup, configuration source	File-based, complex code	XML	Iterative or JSON
Configuration and setup of the experiment	File-based, complex code	XML	Iterative or JSON
Dataset distribution	n/a	Uniform or manual	Yes, embedded
Customisable data distribution strategies	No	No	Yes
Customisable learning strategies	No	Yes	No, pipeline
Collected metrics and data	Time, logs	Accuracy, time, transfer	Multiple quality metrics, time, transfer, logs
Results format	Custom app logs	Collected metrics in unstructured text	Collected metrics structured in JSON format
Entry point to the system application	Command line	Local GUI application or script	Web, via API
User-friendly	No	No	Simple API exposed
Complexity of use	Advanced	Hard	Easy

4.3.2 Execution pipeline

In order to monitor the processing of the algorithm in terms of multiple factors, the monitoring program needs to control the execution of the algorithm; otherwise, data statistics collected during experiments could be disturbed, incomplete or unreliable. Hence, running a computer program with an algorithm implementation on a specific platform involves following a given pipeline.

The execution plan (pipeline) of the common DDM algorithm working in the model with the central coordinator is summarised by the state diagram presented in Fig. 4.14. The state machine starts its life cycle in the local execution node, with a local algorithm performing processing. The standard path is then to process local models in the global state. However, to simulate a one-node (non-distributed) local algorithm, it can finish processing in the next state as well. In this type of scenario, the local model is the only one, and is treated as the global model when collecting results. Nevertheless, following the standard path, at this point of the global processing state, simple DDM methods have already generated a final global model, and can complete processing. In contrast, when iterative methods are implemented in the DDM approach, they can step into the next local state and update previous local models with the current global one, and hence improve them. Further available steps are the same as before, so the algorithm can complete processing or go to the next global processing iteration. When the evaluated algorithm is divided into stages that are executed on separate computational nodes, the agents of the evaluation platform can maintain distributed execution of the program. When the algorithm does not follow any interface known by the platform, the execution cannot be orchestrated and evaluated adequately in a controlled way.

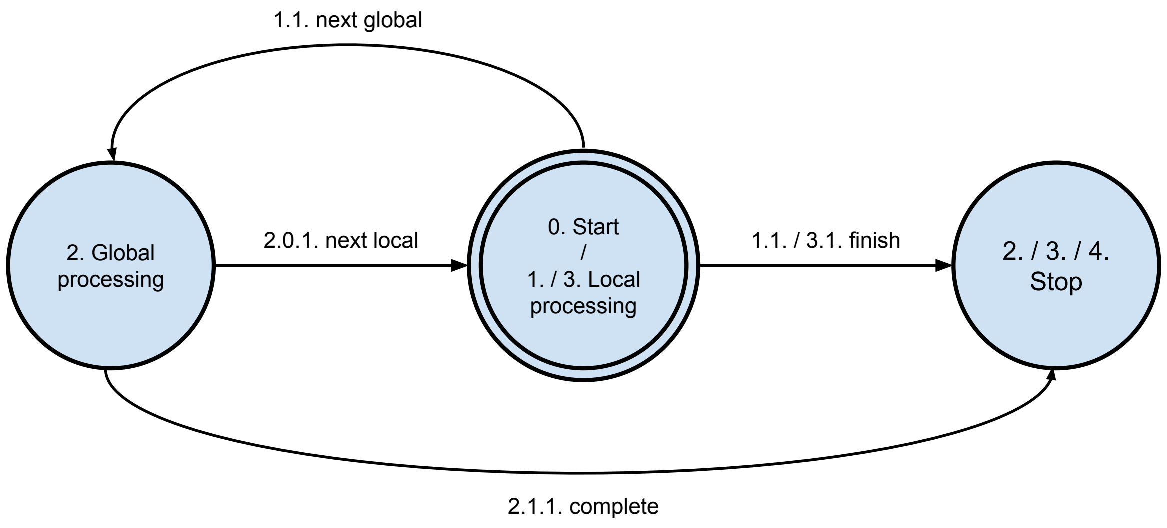


Figure 4.14: State diagram of the DDM pipeline in the model with a central coordinator.

A typical DDM pipeline that involves this state machine is illustrated in Fig. 4.15. It is divided into three vertical parts, with independent local node processing represented on the left and right sides and global coordinator processing in the middle. The state machine shows that some processing stages are optional when the method used is not an iterative algorithm or does not require pre- or post-processing. Examples of the purposes of combining local and global preprocessing include the collection of global aggregates for additional min-max data normalisation, and determining initial algorithm parameters for the next processing stages. At the end of the processing, different actions can be applied depending on the type of algorithm. For a clustering task, local data evaluation is started on the training dataset, while a classification task requires an independent test dataset for better evaluation. Moreover, as global models are already prepared in the last stage, any new samples can be provided for prediction already apart from the algorithm evaluation process.

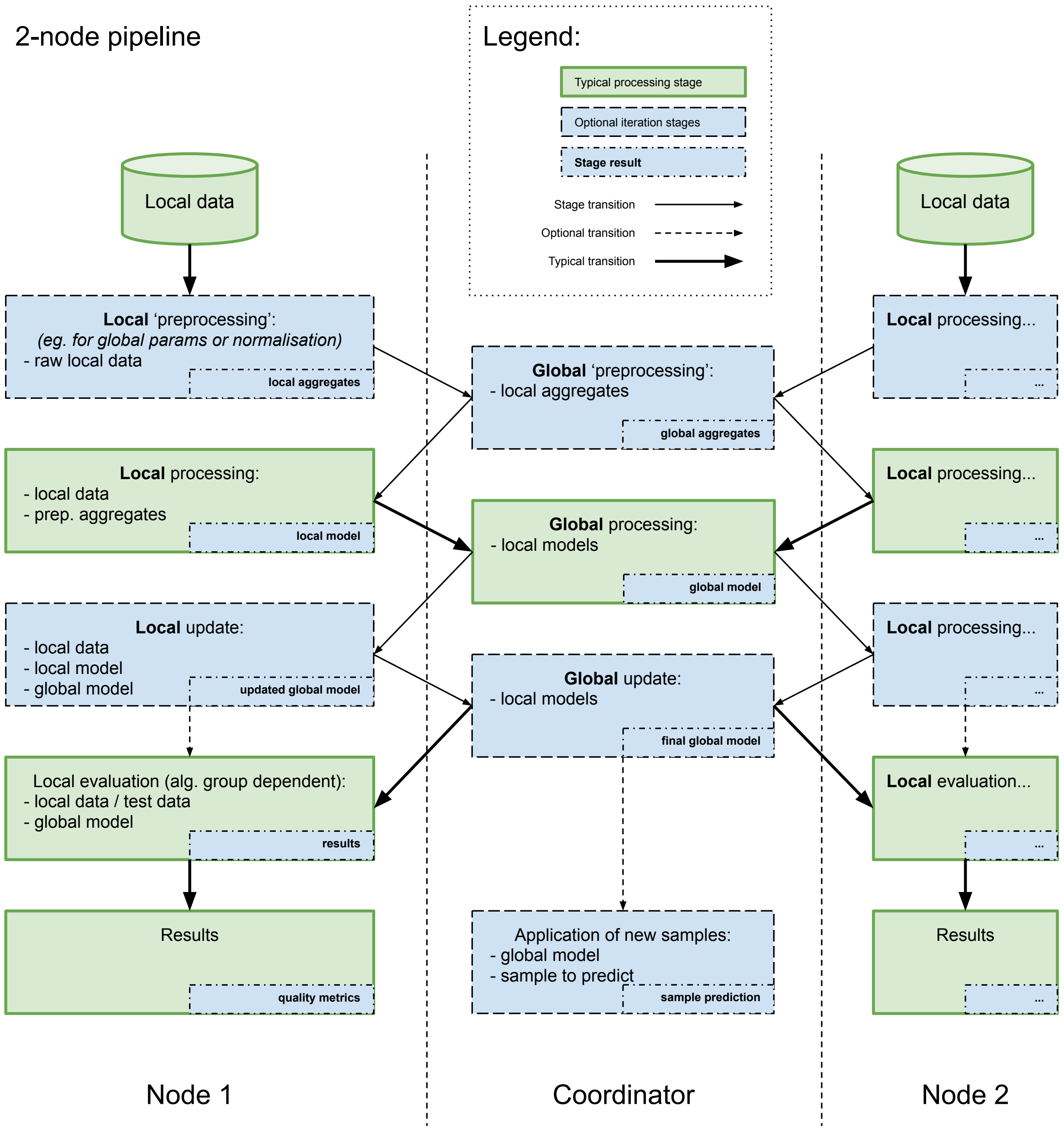
4.3.3 Key components

The whole platform is based on the concept of predefined and user-defined elements. These elements follow the requirements of the framework, which is used within the runtime of an application. We can divide the elements of the system into two main groups. The first group relates to components that are called loaders, as they load the elements required to evaluate algorithms. These required elements include the evaluated algorithms, datasets, partitioning strategies, and the quality and similarity metrics. All of these are depicted in Fig. 4.16, with the relationships between them. The second group refers directly to the execution of an experiment on the platform. In order to run an experiment, the algorithm requires parameterisation, data to be processed, and a place to write the results. Since all of these must be provided, the responsible components are called providers.

Preparation: Loaders

The system has been designed to be as generic as possible, and to be easily pluggable by the user, with its own implementation of the essential elements that are usually data-dependent. Non-predefined algorithms are the most important element in order to avoid the need for rebuilding the whole system for every new algorithm, as mentioned earlier. The same algorithms may be used for different types of data, meaning that custom data similarity metrics can be applied on the platform. This function is most often utilised for data comparisons within algorithm processing, but is also essential for the preparation of data partitions in some partitioning strategies.

According to Fig. 4.16, for each type of platform element (algorithms, datasets, similarity measures, and partitioning strategies) we have a specific type of loader. As already mentioned, algorithms must follow a defined pipeline interface as it is required by the algorithm loader to execute them. Although typical similarity measures such as the Euclidean or cosine measure are often sufficient to compare data during processing, in order to handle the problem of algorithms operating on domain-specific data, custom metrics can be loaded for similarity calcula-



2-node pipeline
Legend:
Typical processing stage
Optional iteration stages
Stage result
Stage transition
Optional transition
Typical transition
Local data
Local 'preprocessing': (eg. for global params or normalisation) - raw local data
local aggregates
Local processing: - local data - prep. aggregates
local model
Local update: - local data - local model - global model
updated global model
Local evaluation (alg. group dependent): - local data / test data - global model
results
Results
quality metrics
Global 'preprocessing': - local aggregates
global aggregates
Global processing: - local models
global model
Global update: - local models
final global model
Application of new samples: - global model - sample to predict
sample prediction
Local data
Local processing...
Local processing...
Local processing...
Local evaluation...
Results
...
Node 1
Coordinator
Node 2

Figure 4.15: A typical, sample DDM pipeline's processing stages with 2 independent nodes and a central coordinator. Green elements with bolded arrows present a standard simple DDM method flow. Dashed blocks are the usual optional processing stages.

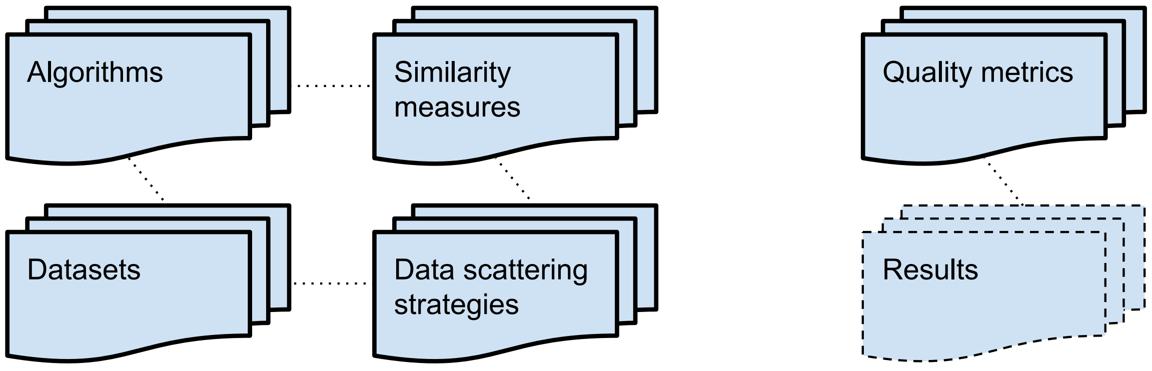


Figure 4.16: Elements of the framework and their relationships. The section with a dashed border is included for the purpose of showing the relationships between the artefacts of the platform.

tion between data samples represented by platform abstraction, as described in the following subsection. The platform cannot limit datasets to the available internal set, as it is limited in the example work [12], and should therefore allow different datasets to be imported in known formats. When loading data with a view to scattering them, it is essential to uniquely identify each data sample among independent locations. This is why, in our platform in the preprocessing step it is optionally possible to generate identifiers when no such attribute exists in the data. The key element is the ability to implement and load a custom partitioning strategy, which also operates on data abstraction.

Execution: Providers

Each method needs to follow certain processing steps and data models in order to supervise the process and collect results that can be used for automatic analysis. We have proposed a simple generic processing model that may be applied to any algorithm by using a wrapper for the execution stages. We can assume this because each DDM algorithm relies on certain blocks of code or processing stages that can be wrapped with the providers of data and parameters without changing the processing flow. More specifically, the sample algorithm may be implemented in isolation from our benchmark platform. However, to run a custom algorithm on the platform, it is only necessary to implement a data structure mapper and several wrapping functions that execute the real implementation of the DDM processing stages. In this section, we describe elements related both to data and processing execution. These elements correspond to specific stages of processing in the platform, such as the data structure, preparation, and the evaluation of the model.

In order to enable the system to operate on datasets that do not follow a specific data format, a standard abstraction has been prepared for the data. This is required for the execution and evaluation of results from generic or user-defined partitioning strategies. A data model abstraction provides object identifier and label attributes and numeric or nominal (by default, a text value) attributes and their types. These data must be provided in the standard way, just as the defined execution parameters for every test case execution together with the chosen similarity measure. These two providers supply inputs for the algorithm. The results collector handles the outputs generated by the algorithms and retrieves predictions for the unique identified samples provided at the results evaluation stage of processing. During execution of the test case in the platform evaluation process, the node agent collects multiple types of statistics, which are important to enable a comparison of algorithms in terms of data transfer load and processing time depending on the phase. IO operations involving the reading of data from disk storage may have different access times, so the system should be able to exclude such times from the comparison. An evaluation time is supposed to be distinguishable because some final models may be too complicated and impact the decision time, which may be unacceptable or constitute another point of comparison between algorithms.

The location of each execution-provider component in the evaluation platform processing is illustrated in Fig. 4.17.

4.3.4 Limitations of the platform

Due to the wide range of different techniques for processing general DDM algorithms, it is impossible to provide a single benchmark platform that can carry out evaluation for every technique, programming language or already existing method as part of a single tool. The current research targets different approaches or methods based on a coordinated server, such as the recently popular FL technique or other iterative processing. However, customisable learning strategies such as collaborative or P2P systems should not be forgotten. The use of various programming languages for solving current DDM problems with different data types should also be addressed.

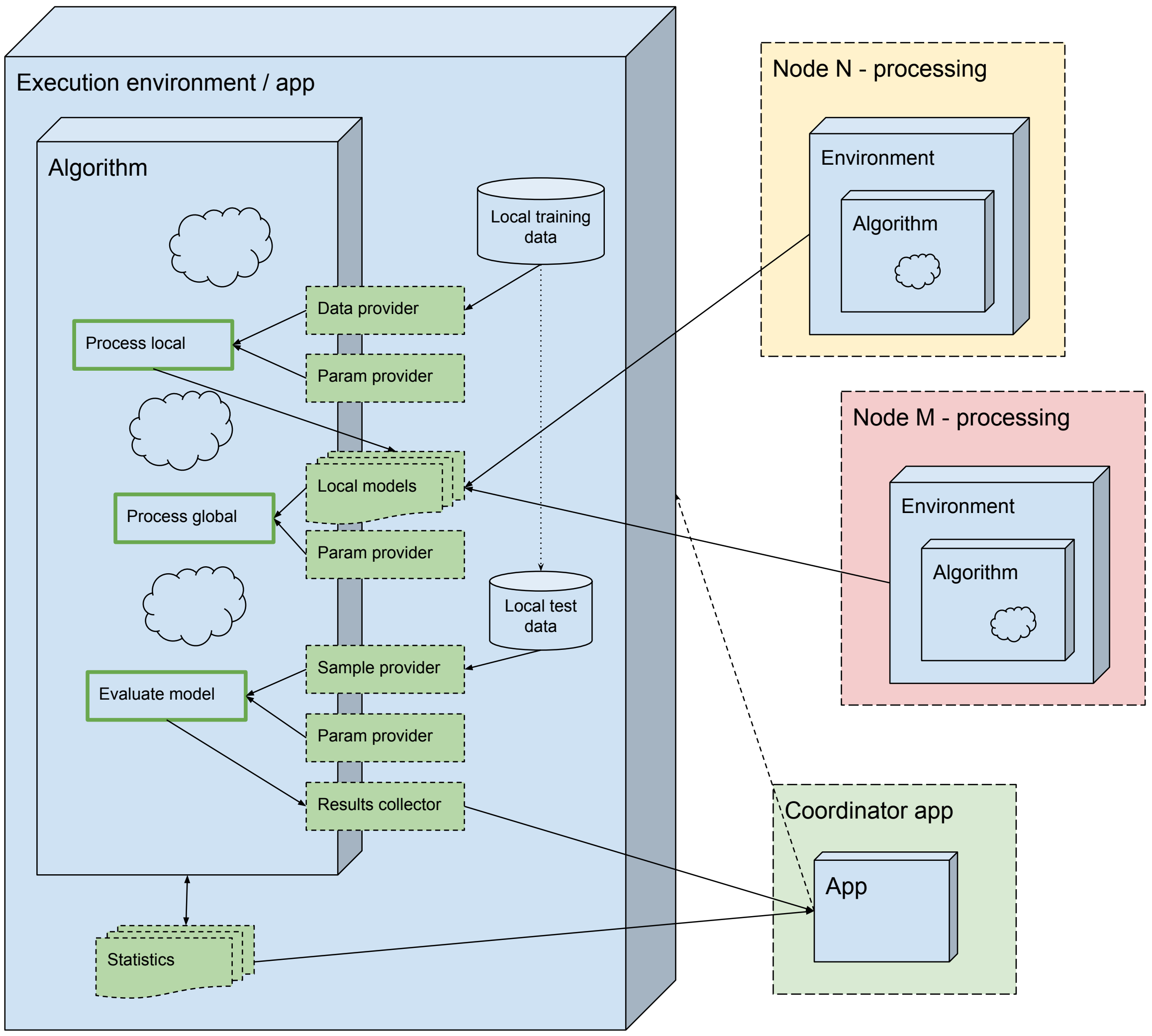


Figure 4.17: Detailed execution environment of the algorithm on a single node. Interfaces of model and execution-provider components are placed in the middle, shown as green areas with a dashed border.

Chapter 5

Experiments

We now introduce the algorithm evaluation process that was performed using different datasets and data partitioning strategies. In our experimental evaluation, the test suites are basically divided into two groups based on the type of algorithm used. We therefore address datasets for classification and clustering algorithms separately. Most of the clustering algorithms evaluated here are non-deterministic, as they typically use a partitioning approach requiring multiple executions for proper evaluation. Hence, each test case was executed multiple times, and we collected processing times and quality information to assess the statistical significance of the results.

The quality measure used for clustering is the ARI, whereas for classification we use the Accuracy for balanced datasets and the F1-score for several other datasets with unbalanced numbers of classes. Besides the impact on the processing time and the differences in transfer load, we are most interested in the deterioration in quality for a typical uniform distribution. To examine algorithms in these terms with differently distributed data and to compare the impact on the results, we need to have reference classes and clusters. We therefore select an external quality measure to analyse the clustering results. Although some of the evaluated datasets consist of small, unbalanced clusters, where the AMI measure would be a better choice, we decided to use ARI for two reasons: firstly, we are more interested in the differences in the results when comparing how the algorithm works with different data distributions, rather than achieving higher values for the results; and secondly, due to the large number of test cases considered in the experiments, the significantly slower computation of AMI quality measures would unnecessarily increase the time required to collect meaningful results for our current evaluation. We aim to compare the results for the same measure, obtained after execution of the algorithm for different types of data partitioning, rather than to enhance the quality of the algorithm. However, as the computational complexity of quality measures for classification is linear, we collect multiple measures for each execution. Full results are presented in the charts attached in the Attachments section 7.

In order to verify the validity of the proposed partitioning strategies, we carried out several experiments. To illustrate the impact on the different aspects of the results, we ran selected distributed algorithms on datasets that were independently partitioned using multiple strategies. All experiments were carried out on the DDM-PS-Eval platform, meaning that all of the proposed methods were ready to use, and were implemented in the way suggested by the authors following given interfaces. This allowed us to collect multiple types of information, such as the processing time, transfer load, and overall quality.

For the experiments, we chose several datasets (four for classification and four for clustering), and these are described in detail in a later section. We deliberately did not use many partitions, in contrast to the strict FL approach, and the sizes of the datasets were not enormously large, since the primary goal was to explore the impact of non-IID partitioning rather than to add artificial complexity to the problem. Moreover, we did not want to artificially increase the complexity of the processing time by adding more samples or generating more partitions with identical distributions, due to the small number of distinct labels in most datasets. We chose a couple of algorithms in their distributed versions, as explained in the following subsection. The selected strategies were parameterised in multiple ways depending on the dataset. The method of selecting the parameters for partitioning each dataset in each strategy is also described in the next section. For more interesting test cases, we present visualisation charts of dataset characteristics after applying a partitioning strategy with the corresponding results, for ease of interpretation and discussion.

5.1 Evaluation method

The overall algorithm evaluation process usually consists of multiple test suites, since different elements are involved in a single test case. In our evaluation method, we have multiple elements, such as the number of partitions/nodes used, the environment, datasets, specific partitioning strategies, algorithms, number of executions and parameterisation for both the partitioning strategies and the algorithm execution. We can consider each test case as a single permutation of these elements. The sequential selection process of the test elements can be carried out in the organised form shown in Fig. 5.1. This illustration shows the components for a single test case, their mutual relationships, and the way in which the entire evaluation process is constructed.

The tree representation in Fig. 5.1 shows the preparation process of the test suite from the root, consisting of sequential choices of the following elements for each test case. We need to select each nested element for each test case. The tree hierarchy also naturally defines a test suite by gathering different permutations in the three main top components. As we evaluate multiple algorithms, it is natural to divide the data first and then apply many algorithms to them, as implied by the hierarchy of the dataset and algorithm elements. The same applies to the relationship between the environment and dataset, where the test environment is prepared with independent nodes before we scatter data over them.

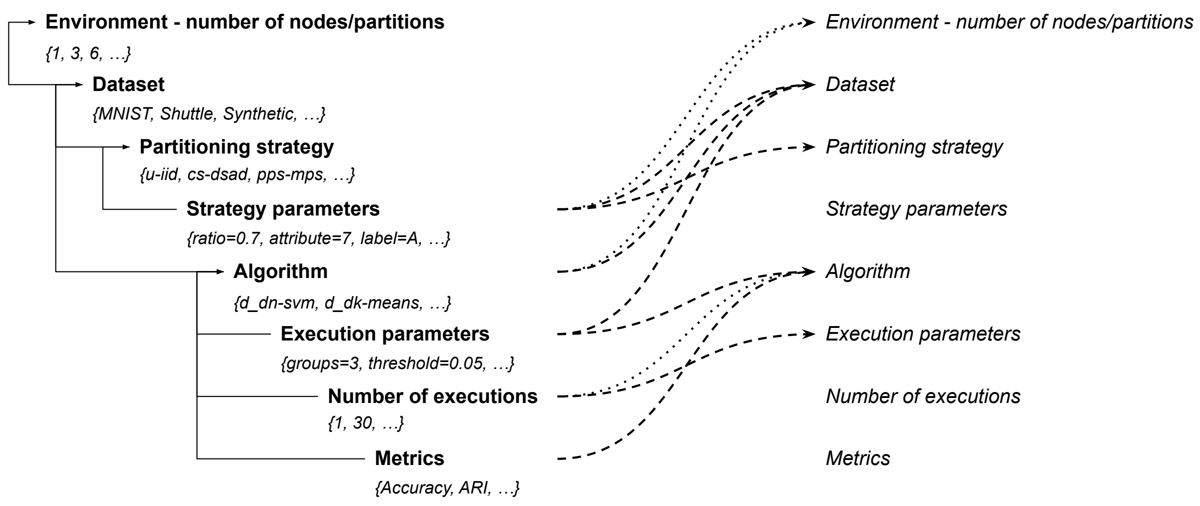


Figure 5.1: Sequence used to choose test case components for the complete evaluation method process (with examples in curly brackets). The relationships between the components and the dependency tree represent this process. Dashed arrows indicate strong relationships, while dotted arrows show minor/common ones.

We distinguish between three main elements in the definition of a test case: the environment, dataset and algorithm. We first choose basic elements such as the number of nodes/partitions, the dataset to be scattered on them, and the algorithm used to process specific data in a particular order. The next selection of nested elements strictly depends on the previous choices, as the partitioning strategy must be able to handle and process the datasets, while the metric depends on the processing results of the algorithm. Both the strategy and the execution of the algorithm must be parameterised, but the latter also depends on the data; for example, for some algorithms such as k-means, we need to specify the target groups to be found. The relationships in the figure above illustrate this clearly. Some relations are apparent; for instance, the parameterising strategy depends on the number of nodes in some cases. However, this is a core element of dataset partitioning, and by definition must work at different scales. The same relation exists for algorithms and concerns the primary distinction between distributed and non-distributed implementations, which work only in a single node environment. The number of executions relates to the determinism of the algorithm: multiple executions are redundant for deterministic implementations, as they always produce the same results. The choice of dataset implies certain strategy parameters, such as the class/label used for concept drift or the attribute used for covariate shift strategies. The use of different datasets enforces the application of a dedicated group of algorithms such as clustering or classification depending on the problem and their specific parameterisation. By going down the list of the elements on the right in Fig. 5.1, we see that each partitioning strategy and algorithm has parameters that need to be set. The number of executions should be considered when the chosen algorithm execution parameters cause the execution of any random-based operations during the processing.

Having proposed multiple partitioning strategies, we wanted to provide a comprehensive evaluation of several algorithms and datasets. We aimed to consider many complex scenarios in a systematic way, in order to identify most of the issues related to non-IID data partitioning. Nevertheless, each element in a given permutation does not necessarily drastically change the outcome of the test case, and this is visible in the results. The differences may be nuanced, such as different parameterisations for non-prior probability shift strategies, which may be irrelevant from the perspective of algorithm processing, as discussed later.

5.1.1 Data distributions

Multiple dataset distributions were evaluated during the experiments, which consisted of the chosen partitioning strategy for each non-IID main category, a different number of target partitions and a partitioning strategy parameterisation that depended on the dataset used. The data partitioning methods are summarised in Table 5.1 in a generalised form, which can be treated as a set of basic test suites for different distributions. We evaluated the proposed methods together with a reference IID distribution. For several strategies, the parameters depend on the chosen attribute or label, or are related to the number of available distinct classes in the data. The parameters used in the table are defined below:

- L is the number of distinct groups/classes in the dataset;
- $\{a\}$ is the attribute chosen for covariate shift partitioning;
- $\{l\}$ is the label chosen for concept drift and concept shift partitioning,
- The parameters for the prior probability shift category, which are used to achieve the multiple data distribution as described in detail in Section 4.1.2, are as follows:
  - Add is the number of additional classes of data samples to add into partitions, where a negative number means the number of classes of all labels that should not be added (i.e. subtracted) from all classes;
  - $EmptyAdd$ is the number of additional classes of data samples to add to the empty partitions, where 0 means the addition of representatives of every class;
  - $MajorityPercent$ is the minimum percentage of class representatives that are to be placed on partitions during partitioning, while the rest are distributed as additional classes and to fill empty partitions;
  - $AdditionalPercent$ is the expected percentage of class representatives to distribute as additional classes, which cannot be higher than $1 - MajorityPercent$, and will be divided into more partitions if necessary to maintain the size of majority groups.

Although the choice of attribute for covariate shift partitioning can be made randomly, this type of experiment would not be well-founded. We therefore analysed the entropy of each dataset and chose the one with the highest value. Some attributes are characterised by strong variety in their values, such as real numerical data. In these cases, we chose the attribute with the highest entropy after discretising the attribute values into a range of 40 equal steps for each, corresponding to the parameter range for the concept drift strategy for consistency. The chosen attribute was passed to the partitioning strategy, while maintaining the other parameters constant for each dataset.

The concept drift and concept shift strategies require the choice of a label for processing. In general, we could choose any label; however, the choice was made based on a class frequency analysis of the data. We assumed that it would affect the partitioning results more as more data samples will be processed, yet partitioning does not rely on quantity only but also characteristics. For a concept shift strategy, this will disrupt the classification results more significantly, while for a concept drift approach, this may provide a wider spread range of data attributes for separation, creating better data partitioning in corner cases.

5.1.2 Algorithms

Most of the clustering algorithms considered in our evaluation fully or partially rely on a partitioning approach, which is unquestionably not the best choice for use with all datasets. Two of them used a hierarchical approach as an ensemble method, and another one was density-based. Thus quality results may be slightly different from the quality obtained in the literature using a single algorithm on the same datasets. We chose the SVM algorithm in its distributed version for evaluation of the classification, as it typically produces excellent results while also being simple to use. At the same time, the processing time depends on the data samples, which will be highlighted in our discussion of the results. Our second choice was a probabilistic classifier, which was developed with the aim of being neutral towards the distribution while operating on independent statistics of data samples.

In addition to the reference results for a uniform IID distribution, we wanted to provide results for non-distributed processing on the dataset used in the evaluation. The aim here was to show convergence between the quality results for the same types of algorithms in both their distributed and non-distributed versions. We therefore wrote and evaluated a simple implementation of the naive Bayes classifier. We also used SVM and k-means implementations from the WEKA [67] tool, which was also utilised for the local processing of our distributed implementations. For the second non-distributed clustering algorithm, we employed BIRCH from the ELKI [2] library, which is an excellent implementation for general data analysis. Obtained results compared in Section 5.2 indicate that the distributed implementations of the algorithms were correct.

We used four distinct clustering algorithms and two simple distributed classifiers for the different data distribution evaluations. The two classifiers were a distributed implementation of naive Bayes, which builds global *a priori* probability statistics based on probabilities collected

Table 5.1: Summary of partitioning strategies for experiments with generalised parameters.

Code	**Main category**	**Strategy**	**Parameterisation**	**Number of partitions**
iid-u	IID	Uniform	n/a	{3, 6, 12}
cs-dao	Covariate shift	Dense and outliers	0.7	

cs-dsad		Diff. statistical attr. distribution	Shift = 0.3
			Splits = 3
			Attribute= $\{a\}$	

pps-s	Prior probability shift	Separated	Add = 0
			EmptyAdd = 1
			MajorityPercent = 0.4
			AdditionalPercent= 0	

pps-ab		All but	Add = -1
			EmptyAdd = $L-1$
			MajorityPercent = 0.5
			AdditionalPercent= 0.1	

pps-ms		Missing some	Add = $-\min(L-1, 3)$
			EmptyAdd = $L-\min(L-1, 3)$
			MajorityPercent = 0.6
			AdditionalPercent= 0.1	

pps-mps		Most plus some	Add = $\min(L-2, 2)$ || $\min(L-1, 2)$
			EmptyAdd = $\max(1, L-2)$
			MajorityPercent = 0.8
			AdditionalPercent= 0.05	

pps-mpa		Most plus all	Add = L
			EmptyAdd = 0
			MajorityPercent = 0.6
			AdditionalPercent= 0.05	

cd-sba	Concept drift	Split by attribute	Drifts = 3
			Ranges = 40
			Label= $\{l\}$	

cs-cl	Concept shift	Artificial label	Shifts = 2
			Label= $\{l\}$	

u-ul	Unbalancedness	Unbalanced labels	Ratio = 0.1
			Threshold = 2
			Proportional= 0	

from local nodes, and the naive distributed SVM approach described in [56], tentatively named DN-SVM. In the latter approach, the final SVM model was trained on locally found support vectors sent from local nodes, which were treated as global training data. The clustering algorithms were DK-means [31], Opt-DKM [49], a lightweight clustering technique [5] tentatively named LCT, and a modified version of the BIRCH algorithm [71], in which a partitioning approach was used for the final clustering. This modification was made by combining the agglomerative process in BIRCH with a partitioning method. BIRCH clusters the data at the local nodes and sends the cluster centroids to the global node, where the same operation is performed by treating the centroids as data samples to be clustered. Finally, global clustering is done by assigning samples to the closest global centroid.

A summary of the implementations described above is presented in Table 5.2. Each algorithm, whether executing on single or multiple nodes, was wrapped into the DDM-PS-Eval processing interfaces model. This allowed us to collect comparable results in a structured form, which could then be easily visualised via a semi-automatic process using spreadsheets with dedicated formulas and scripts to present the values in the form of charts. We intend to incorporate this visualisation process for the results into the platform in the future.

With regard to the parameterisation when executing the algorithms, we used the same parameters for configuration. In general, such an approach would be wrong, as correct parameter adjustments can produce better quality results and allow for the avoidance of pathological situations, such as merging too many clusters due to a high epsilon threshold. Nevertheless, we decided to use a common setup for each algorithm due to the wide variety of test cases mentioned at the beginning of Section 5.1, and in order to compare different partitioning schemes. A precise data distribution scheme is typically unknown in the real world, so parameters cannot be adjusted perfectly. The parameter values are listed in Table 5.2 next to each algorithm for completeness.

5.1.3 Datasets

In our experiments, we assumed that each dataset had already been preprocessed, cleaned and prepared for evaluation. All of these steps are important to achieve good results; however, in the real world, we also have to face unexpected situations, and thus it is better to be prepared for both cases. Our focus was also on uneven data distributions, rather than data preparation. The datasets chosen for evaluation contained primarily numerical data, with several categorical attributes. These were non-textual popular MNIST sets, several well-known datasets from the UCI Machine Learning Repository, and synthetic generated data.

We used the PCA algorithm to preprocess and reduce the dimensionality of the MNIST and FMNIST data from 28x28 to 28 attributes, to avoid applying partitioning strategies to raw pixel values. Due to this preprocessing step, our quality results diverge from those in the literature, but are coherent with reference values obtained from execution on a single node and in the distributed version with a uniform data distribution.

Table 5.2: Summary of the algorithms used in the experiments.

Code	**Algorithm**	**Type**	**Purpose**	**Distri-buted impl.**	**Execution parameterisation**
s_naive-bayes	Naive Bayes	Probabilistic	Classification	no	none
s_svm	SVM	SVM	Classification	no	kernel: 'rbf'
s_k-means	K-means	Partitioning	Clustering	no	groups = L iterations = 20
s_birch	BIRCH	Hierarchical	Clustering	no	branching_factor = 50 threshold = 0.01 groups(maxleaves) = L
d_naive-bayes	Distributed naive Bayes	Probabilistic	Classification	yes	none
d_dn-svm	Distributed naive SVM	SVM	Classification	yes	kernel: 'rbf'
d_dk-means	DK-means	Partitioning	Clustering	yes	groups = L iterations = 20 init_kmeans_method = k-means++
d_opt-dkm	Opt-DKM	Partition-density based ensemble	Clustering	yes	epsilon = 0.002 minKGroups = true groups = L iterations = 20 init_kmeans_method = k-means++
d_lct	LCT	Partitioning-hierarchical ensemble	Clustering	yes	b = 2 groups = L iterations = 20 init_kmeans_method = k-means++
d_dp-birch	Distributed agglomerative BIRCH	Hierarchical-partitioning ensemble	Clustering	yes	branching_factor = 50 threshold = 0.01 g_threshold = 0.01 groups (maxleaves) = L g_groups = L iterations = 20

With regard to the synthetic data, we performed multidimensional numeric dataset sampling with a normal distribution to obtain fairly well-separated Gaussians, in a similar way to that described, for example, by the authors of [32]. The first was generated as two-dimensional points, arranged in Gaussian clusters with overlapping outliers, with two groups colliding in order to avoid perfect clustering results for the tested algorithms, as they use partitioning methods internally (in this case, k-means). The second dataset was generated in four-dimensional numerical space to increase the transfer load and computational complexity, which translates into a longer processing time. Using this generation strategy, we obtained labelled data to use for reference clustering.

One of the datasets considered here was introduced together with the CURE algorithm in [22] as "Dataset 1" but without noise. We chose this dataset as it was developed for a hierarchical clustering algorithm and contained mostly Gaussian shapes of clusters for partitioning. It thus posed two challenges for the clustering mechanism incorporated in our ensemble of distributed algorithms, which was expected to produce interesting results. However, the original CURE dataset is small, and contains only 2000 samples. We therefore expanded it by adding 50000 samples for each cluster, where the data were generated using a normal distribution based on the mean of each cluster and a variance analysis. The original data and the results of the expansion operation are illustrated in Fig. 5.2.

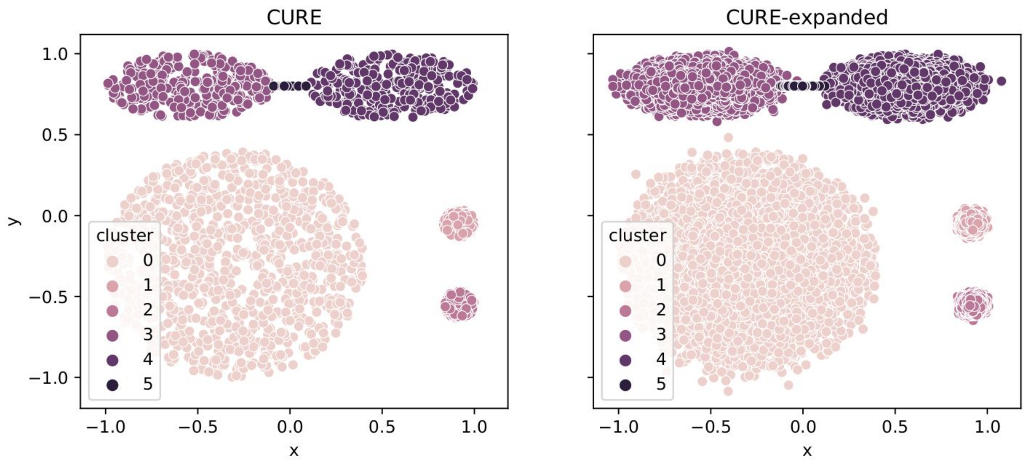


Figure 5.2: Visualisation of (a) CURE Dataset 1; (b) the expanded CURE Dataset 1.

A similar situation arose for the Diamond9 dataset, which was introduced in [61]. The amount of samples was small, and the shapes of the clusters allow them to be clustered by partitioning, density-based or hierarchical algorithms. We therefore performed the same type of data expansion, slightly changing the shape due to the normal distribution. The original data and the results of the expansion operation are illustrated in Fig. 5.3, in the same way as for the previous data.

We also included datasets from the UCI repository containing categorical attributes in the evaluation of the classifiers. With regard to the data distribution, only the covariate shift ap-

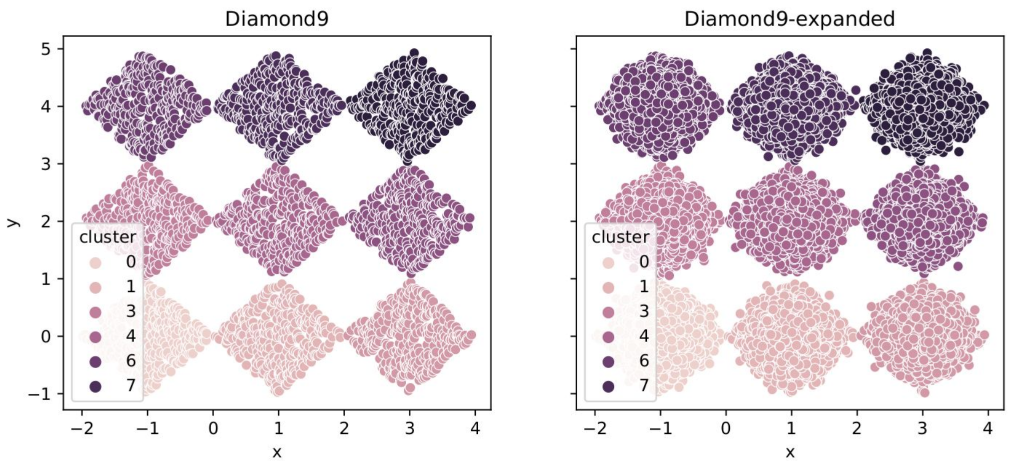


Figure 5.3: Visualisation of (a) the Diamond9 dataset; (b) the expanded Diamond9 dataset.

proach of the proposed partitioning strategies takes into account the attributes in processing. As described earlier, the 'cs-dsad' strategy could work by counting the attribute values, while the 'cs-dao' strategy requires numerical attributes or distance measures for the data. The same issue applies to the SVM algorithm, which uses the Euclidean distance to find separating hyperplanes. The DDM-PS-Eval platform used in our evaluation is able to load and provide custom similarity metrics, and to handle nominal attributes, if the algorithm uses the provided distance function. However, in order to standardise the experiment, we decided to use the same numerical metric for each algorithm evaluated, which was the Euclidean distance. In view of this, each categorical attribute in the data underwent a vectorisation operation at the preprocessing stage; however, this vectorisation was not sophisticated, and did not involve an analysis of data semantics but allowed nominal data to be used with numerical similarity metrics to convert the type of attribute. We transformed nominal values for each attribute separately, mapping distinct values into the subsequent natural numbers. This mechanism was already implemented in the evaluation platform. Following this, we were able to apply the 'cs-dao' strategy to each dataset and to use a common distance metric. In this work, we only highlight the issue of missing similarity metrics in different works, as mentioned in Section 4.3, and focus on other aspects of the evaluation. Furthermore, our platform provides the possibility of dealing with this issue in a generic way for newly developed algorithms, as set out in the description of the platform.

All of the datasets and their original features are summarised in Table 5.3. We divided the datasets into those used for classification and those for clustering.

As a result of each test case execution, we obtained a global model, which was then applied to the data. For clustering, we used the entire data on each partition to achieve global clustering, while for classification, we needed a verification set to examine the global classification model that had been built. For those datasets that did not provide a dedicated testing set, we divided the entire dataset using a ratio of 60 : 40 to create training and testing data samples. We did not apply a cross-validation method, as this would have conflicted with the partitioning strategy results;

in a scenario with cross-validation, it would be necessary to execute separate partitioning for each submodel, which would generate more permutations for evaluation, thus preventing the possibility of analysing the combined results for a specific data distribution.

Table 5.3: Summary of primary raw datasets.

Dataset	**Problem type**	**Data type**	**Instances (train + test)**	**Attrs.**	**Classes /Groups**
MNIST[1]	Classification	Numerical (image pixels)	60000 + 10000	784	10
FMNIST[2]	Classification	Numerical (image pixels)	60000 + 10000	784	10
Shuttle[3]	Classification	Numerical (integer)	43500 + 14500	9	7
Adult[4]	Classification	Categorical + numerical (integer)	32561 + 16281	14	2
CURE-expanded[5]	Clustering	Numerical (real)	302000	2	6
Diamond-expanded[6]	Clustering	Numerical (real)	453000	2	9
Synthetic 7g	Clustering	Numerical (real)	100000	2	7
Synthetic 12g	Clustering	Numerical (real)	1200000	4	12

5.2 Non-distributed execution

In this section, we present the results of non-distributed processing obtained for each of the datasets. The results are illustrated in the form of charts in Figs. 5.4 and 5.5, together with the maximum values obtained from the execution of the distributed algorithms on three nodes, to demonstrate the similarities. From an examination of pairs of algorithm types, we can see that the classification results are comparable. From the clustering results, we can see data-dependent and mixed results from the ensemble methods, which will be discussed further. The exception in the results is the LCT implementation, which gave lower values, as it is generally too greedy at the aggregation stage and merges too many clusters, generating them in a faulty way.

[1]https://www.kaggle.com/datasets/oddrationale/mnist-in-csv

[2]https://www.kaggle.com/datasets/zalando-research/fashionmnist

[3]https://archive.ics.uci.edu/ml/datasets/Statlog+(Shuttle)

[4]https://archive.ics.uci.edu/ml/datasets/adult

[5]https://github.com/deric/clustering-benchmark/blob/master/src/main/resources/datasets/artificial/cure-t1-2000n-2D.arff

[6]https://github.com/deric/clustering-benchmark/blob/master/src/main/resources/datasets/artificial/diamond9.arff

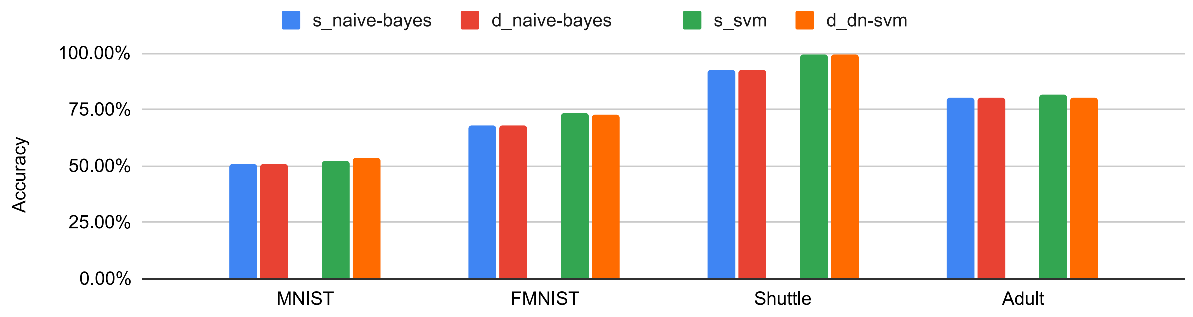


Figure 5.4: Results for single-node classification and comparison with the maximum quality results for distributed algorithms executed on three nodes.

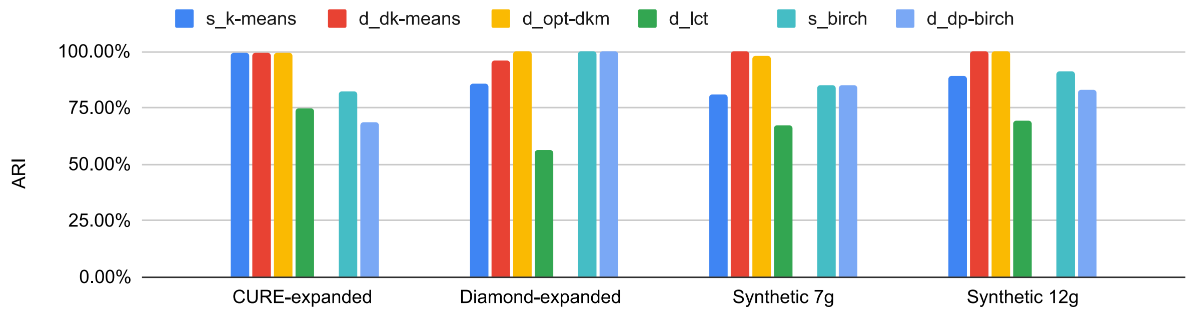


Figure 5.5: Results for single-node clustering and comparison with the maximum quality results for distributed algorithms executed on three nodes.

5.3 Evaluation results

5.3.1 Quality results

In order to enable a better impact analysis, we divide the impacts of the different distributions on the results into several categories. The difference in the average quality results for the partitioning strategies is compared to a uniform distribution with the 'iid-u' strategy for each algorithm. A negative impact can be assigned to one of three negative categories depending on the percentage difference, as low ($< 5\%$), medium ($< 10\%$), or high ($\geq 10\%$) impact. There are two other categories: one where the difference range is $[-2\%, 2\%]$, which represents no noticeable impact on the results, and a positive impact category ($< -2\%$), where we obtain better results than using a uniform data distribution. Tables 5.5, 5.6, and 5.7 show the results, and we also use a heat map to highlight these categories. Green indicates a positive impact, while no colour is used for the negligible range. We use yellow, pink, and red for low, medium and high negative impact, respectively.

Classification algorithms

From the results for the naive Bayes classifier in Fig. 7.2, we see that there is almost no impact, regardless of the partitioning used. Some algorithms are naturally neutral to the distribution, as they operate on independent statistics that can be aggregated without information loss; this is the case for naive Bayes, which processing was only affected by concept shift partitioning. This classifier is non-fuzzy and incorrectly assigned artificial classes to test data samples degrading the final quality. This neutrality is a desirable aim for the design of all algorithms, but this behaviour is not standard, as shown by the other results. The differences in the resulting impact using the 'cs-cl' strategy with different datasets arise from the data sizes and the formula used to prepare the artificial classes in the data. As we used the same formula, we could observe the differences in the results from the same datasets when divided into various numbers of partitions. To analyse the origins of these differences, we can count the number of artificial classes in the data created using the MNIST dataset as an example; where the impact is fading with more partitions. In our experiments, we assigned the label '1' to the 6742 samples that were the most frequent of all 60000 in the training set. Based on the formula in Equation 4.7, we can count the number of artificial classes and compare their impact on the 'iid-u' partitioning accuracy from the summarised results in Table 5.4. More *"shifts"* generally decrease the probability of the presence of the original class, causing lower accuracy. On the other hand, an increasing number of partitions limits the possible number of artificial samples present in the experiment. At the same time, it keeps the equal number of the origin class on every partition in number $\frac{1}{N} \cdot |D_c|$.

Table 5.4: Comparison of accuracy results for the naive Bayes classifier on the MNIST dataset, divided into 12 partitions using the 'cs-cl' strategy for class '1', and for different $shift$ values.

$shift$ **param**	**Artificial samples**	**Accuracy**	**Impact**
0 - 'iid-u'	*0*	*50.47%*	*n/a*
2	1123	50.47%	0%
6	3371	50.19%	0.28%
9	5056	48.67%	1.8%
11	6180	43.17%	7.3%

The SVM classifier results are strongly affected by various distributions as it operates on the final support vectors found in the global scope. Global support vectors do not always reflect the entire dataset's characteristics, as the global model is built based on partial information from local models. Failed predictions after training on data partitioned with the 'cs-cl' strategy have the same origin as for naive Bayes. Moreover, the family of SVM algorithms are binary classifiers, and hence multiclass classification requires the production of extra weak classifiers that participate in voting. However, in our experiments, these were a minority of all the weak classifiers built, due to a low value of the $shift$ parameter. This is why the origin of the incorrect predictions lies in the higher number of samples with artificial classes to separate from the real ones. The impact is reduced for more partitions, for the same reason as described above and illustrated in Table 5.4.

The highest number of wrong SVM predictions is seen for the prior probability shift partitioning strategies. This arises from the overfitted local processing results, based on which the global model is built. The correct support vectors are also missing when there are missing samples on the local site. A lack of information is propagated to the global model, which is then built using incomplete data. This type of situation occurs for partitioning strategies with missing classes, such as 'pps-s', 'pps-ab', 'pps-ms', and 'pps-mps', while the impact is lower for the 'pps-mpa' strategy. In the last strategy, there are no missing labels; instead, there are insufficient data samples from minority classes to prepare comprehensive local models for some test cases.

Data partitioning strategies based on attribute distribution affect the SVM classifier significantly less than those operating on data distribution per class. However, separating data using similar data samples, as in the 'cd-sba' or 'cs-dao' strategies, has implications in terms of preparing worse local models. The differences in distribution obtained for 'cd-sba' on the Shuttle dataset are illustrated in Fig. 5.6. In this test case, our concept drift partitioning strategy considered the sequence $[8,4,7,3,1,5,2,6,0]$ of attributes based on the entropy analysis, and performed *drifting* by attribute '0'. The data sample shift across partitions led to shifted values of the separating hyperplanes for the independent local models built to predict the same class on every partition, which made the global model predictions worse.

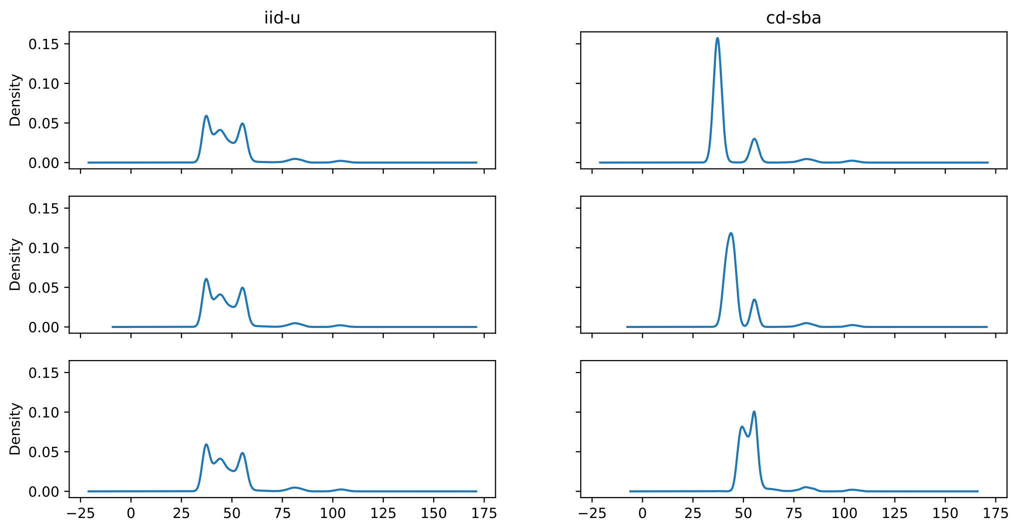


Figure 5.6: Comparison of the density of data samples for attribute '0' with class '1' from the Shuttle dataset, for the 'iid-u' and 'cd-sba' data distributions with three partitions.

For several test cases for the 'pps-s' strategy in which the number of classes was less than the number of partition nodes, SVM could not provide any results, as no support vectors were found and hence data separation was impossible. The same situation arose for the Adult dataset using the 'pps-mps' strategy, where data distribution with a single missing class for binary classification reduces it to the 'pps-s' distribution. Similar strategy convergence was found for 'pps-ab', which in this case is equivalent to 'pps-ms' with a difference in the empty partition fill parameters. Hence, for parameterisation that assumes class separation among independent nodes, the standard SVM predictor will always fail to process, with 0 accuracy and *NaN* F-score, as it cannot perform any prediction. We mark these cells in Table 5.5 as 'n/a'. The differences in the label distribution for 'pps-s' and 'pps-ms' are presented in Fig. 5.7. From the results, we see that when the number of labels exceeds the number of partitions, a uniform distribution of every label representative on empty partitions gives better classification of data. This demonstrates the need for improving some minor partitioning aspects of the partitioning strategy algorithm, such as distributing low amounts of data or labels into higher numbers of partitions. This is relevant to avoid such facilitation for algorithms and to prepare better test cases.

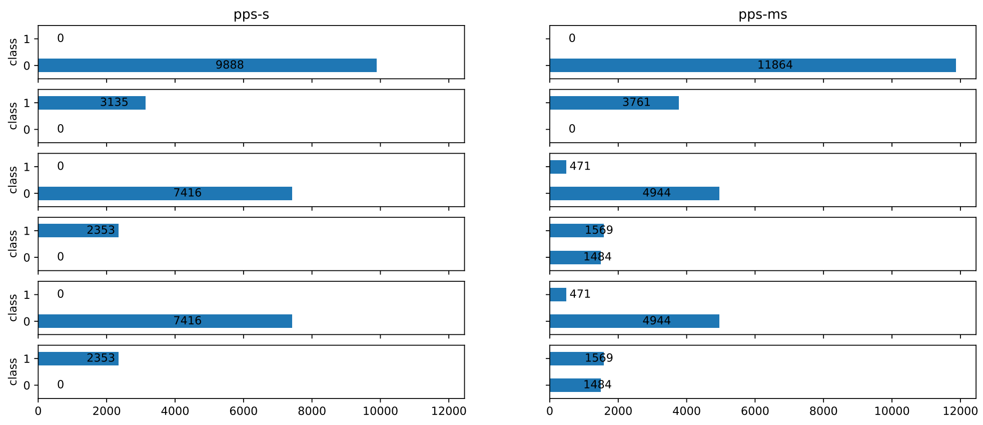


Figure 5.7: Comparison of the data sample label distribution for the Adult dataset, using the 'pps-s' and 'pps-ms' strategies with six partitions.

The overall conclusion from an analysis of this pair of algorithms is that a covariate shift strategy at a larger scale will always affect non-fuzzy classifiers. Moreover, the prior probability shift approach strongly impacts the prediction when the global model is built from incomplete local models, meaning that more sophisticated methods are required when merging models. Nevertheless, it is crucial to evaluate each distributed algorithm using multiple partitioning strategies to reveal its weaknesses or neutral behaviour, as illustrated by the discussion above. We can even conclude that for the classification algorithms, regarding the quality of the results, an adequate minimal test coverage suite would require an evaluation of the concept shift and prior probability shift approaches only instead of the one illustrated previously in Fig. 4.3.

Table 5.5: Differences in the classification quality results for partitioning strategies, compared to the uniform distribution with the 'iid-u' strategy, with a heat map showing five categories of partitioning impact.

	strategy	**3 nodes**				**6 nodes**				**12 nodes**			
algorithm	**dataset**	MNIST	FMNIST	Shuttle	Adult	MNIST	FMNIST	Shuttle	Adult	MNIST	FMNIST	Shuttle	Adult
d_dn-svm	cs-dao	1.02%	1.44%	5.49%	.41%	-.15%	1.43%	2.43%	.41%	.31%	.64%	-4.42%	.24%
	cs-dsad	.52%	.27%	1.42%	-.02%	.12%	.04%	2.11%	-.02%	.24%	.32%	2.03%	-.05%
	pps-s	4.97%	24.29%	.89%	n/a	11.42%	32.49%	51.43%	n/a	n/a	n/a	n/a	n/a
	pps-ab	1.03%	2.73%	.43%	3.28%	-.02%	.27%	.94%	2.30%	.21%	.06%	.31%	2.70%
	pps-ms	1.70%	1.60%	2.02%	1.43%	.55%	.18%	5.54%	1.43%	.29%	.28%	.95%	-.51%
	pps-mps	2.74%	16.04%	11.46%	n/a	4.88%	14.32%	8.59%	n/a	-.05%	-.24%	-2.49%	n/a
	pps-mpa	.52%	1.79%	.33%	2.36%	-.11%	.73%	1.32%	2.36%	-.47%	.54%	.40%	-.07%
	cd-sba	.27%	.13%	8.72%	7.48%	.62%	-.14%	9.32%	7.48%	.57%	1.34%	7.62%	6.27%
	cs-cl	7.66%	1.77%	20.98%	57.31%	.17%	-.63%	-2.96%	57.31%	.29%	-.12%	.43%	16.09%
	u-ul	.50%	.00%	.21%	-.11%	.25%	-.30%	.29%	-.11%	-.09%	-.05%	.03%	-.53%
d_naive-bayes	cs-dao	.00%	.00%	.00%	.00%	.00%	.00%	.00%	.00%	.00%	.00%	.00%	.00%
	cs-dsad	.00%	.00%	.00%	.00%	.00%	.00%	.00%	.00%	.00%	.00%	.00%	.00%
	pps-s	.00%	.00%	.00%	.00%	.00%	.00%	.00%	.00%	.00%	.00%	.00%	.00%
	pps-ab	.00%	.00%	.00%	.00%	.00%	.00%	.00%	.00%	.00%	.00%	.00%	.00%
	pps-ms	.00%	.00%	.00%	.00%	.00%	.00%	.00%	.00%	.00%	.00%	.00%	.00%
	pps-mps	.00%	.00%	.00%	.00%	.00%	.00%	.00%	.00%	.00%	.00%	.00%	.00%
	pps-mpa	.00%	.00%	.00%	.00%	.00%	.00%	.00%	.00%	.00%	.00%	.00%	.00%
	cd-sba	.00%	.00%	.00%	.00%	.00%	.00%	.00%	.00%	.00%	.00%	.00%	.00%
	cs-cl	.45%	3.04%	12.58%	10.80%	.02%	.13%	13.86%	10.80%	.00%	.17%	14.18%	.31%
	u-ul	.00%	.00%	.00%	.00%	.00%	.00%	.00%	.00%	.00%	.00%	.00%	.00%

Clustering algorithms

We first note that the evaluation of non-deterministic algorithms requires multiple executions. Otherwise, an analysis or comparison of the maximum obtained results would lead to false conclusions, and we could never be sure of the final result; hence, most discussions are based on an average score from multiple executions. However, we do refer to the best possible results in specific situations in order to articulate some of the different weaknesses of the algorithms.

From the results in Tables 5.6 and 5.7, we see differences between the algorithms irrespective of the data used. For non-deterministic algorithms, initialisation usually plays a crucial role and affects the final results, and this is evident for the DK-means algorithm. The partitioning strategy influences the initialisation process, and typically leads to wrong global results. However, the same results shown in the summarised charts in Fig. 7.3 indicate that the maximum possible values are achievable. This suggests that in the design of an algorithm, more attention should be paid to better local model preparation to avoid global errors. On the other hand, the Opt-DKM density-based approach to global processing correctly handles local models for most strategies without intervention in the local model preparation. A specific data distribution helps to build a local model structure for a hierarchical type of algorithm, and therefore gives better results than for a uniform data distribution. However, the use of an agglomerative hierarchical approach in the second global stage of processing, unlike in the previous hybrid approach, reduces the quality for higher numbers of partitions as the merging operation in the LCT algorithm fails.

Pure partitioning and density-based ensembles The clustering results show how the mean and maximum possible values of the clustering quality may vary depending on the data partitioning. It is clear that even the local use of k-means++ initialisation [6] did not help in terms of avoiding the impact of the different data partitioning methods on the global results. It is also interesting that in most cases, a uniform data distribution made it difficult to obtain the best results, which was not expected at the outset. Natural concentrations of similar data based on distance measures are helpful for density-based algorithms, as these naturally search for dense clusters, although this may be problematic for pure partitioning methods, as they search for a fixed number of partitions and may initially divide similar data into separate clusters that may never be corrected in further iterations or on the global stage.

We now analyse the maximum possible values obtained by the DK-means algorithm on the dataset, as illustrated by the charts in Fig. 7.3. In this case, only single partitioning in the 'pps-ms' strategy gives worse results for three partitions on the Synthetic 12g dataset. The data sample label distribution for this test case is shown in Fig. 5.8. Again, the initialisation issue was the root cause of this poor quality outcome from the algorithm. Data samples of multiple class representatives were missing, and the number of present samples was more significant than for the other evaluated datasets, which led to incorrect initial clustering. Moreover, partitioning at the global stage could not fix the small number of incorrect local centroids. At the same time, the density-based approach worked correctly as designed and dealt well with the same distribution, but failed with the ’pps-mpa’ strategy distribution, as also illustrated in Fig. 5.8. Based on this, we can assume that the possible significant deterioration in results is due to the wrong initialisation, which is a crucial aspect. This is particularly true in the distributed environment, where even verified initialisation methods such as k-means++ sampling fail as they are not designed for non-IID data distributions.

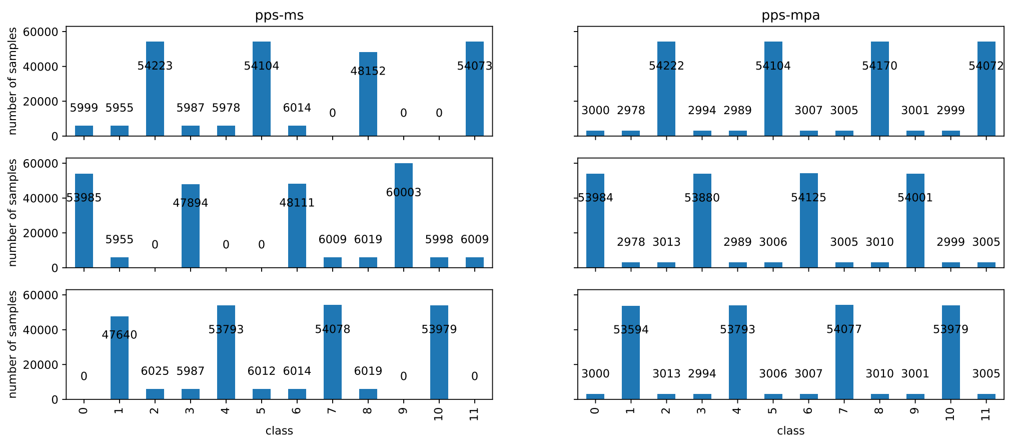


Figure 5.8: Comparison of the data sample label distributions for the Synthetic 12g dataset, using the ’pps-ms’ and ’pps-mpa’ strategies with six partitions.

Different outcomes for the maximum possible results are seen for the Opt-DKM algorithm, where the density feature did not work for several test cases. The worse results arise from the excess number of clusters found, due to the use of the *minKGroups* parameter in our evaluation. Nevertheless, the overall average results are promising, as the various strategies influence them in only half as many test cases compared to the DK-means algorithm. This suggests that the density-based assumptions achieved the objectives, which primarily involved fixing the issue of incorrect local initialisation affecting global processing.

The results of the ’pps-s’ strategy are particularly noteworthy, as both algorithms failed on the CURE-expanded dataset regardless of the number of partitions. The full separation of clusters among partitions to simulate the scenario of already clustered data within the data partition boundaries is problematic for DK-means. The algorithm tries to find a specified, fixed number of groups inside the partition where fewer groups are available. The global partitioning then fails to find global centroids based on the incorrect local ones. For Opt-DKM, the issue is not associated with the higher number of local groups found but with an inability to merge local models for one significantly wider group and additionally with a lower density of samples, as illustrated in Fig. 5.9 with the centroids found by both algorithms.

For DK-means, solving this issue depends on initialisation, as good clustering is possible, confirmed by the maximum quality results obtained in experiments. In Opt-DKM, more inves-

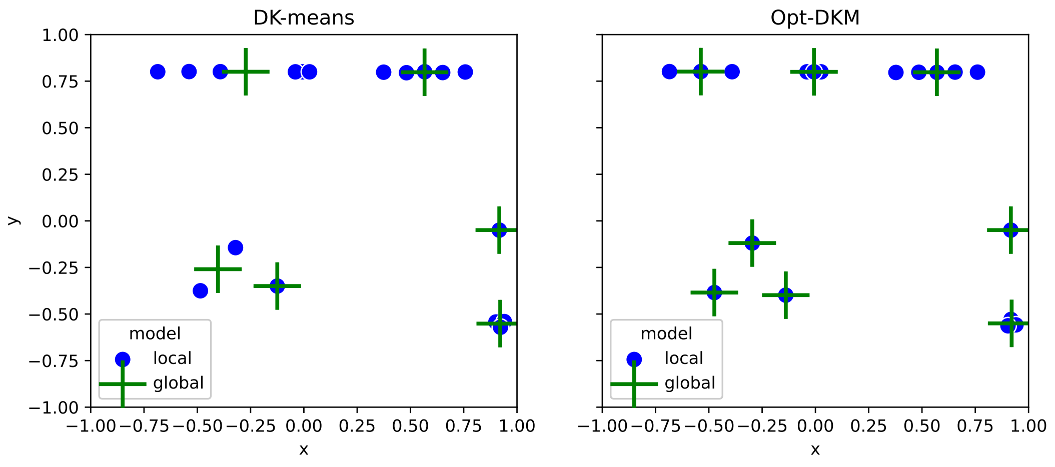


Figure 5.9: Examples of imperfect DK-means and Opt-DKM algorithm models found for the expanded CURE dataset, divided into three partitions using the 'pps-s' strategy.

tigation is needed in terms of approximating the ε' parameter in global OPTICS, since the same issue relating to the division of cluster '5' shown in Fig. 5.2 into more groups occurs for the 'cd-sba' strategy. Hence, strong separation of the data samples group across partitions done by the partitioning strategy impacts this concrete density approach more for data with divided clusters consisting of low boundary ranges of attribute values. Since the estimation of ε' is based mainly on the cluster statistics found, low values make it impossible to merge those clusters.

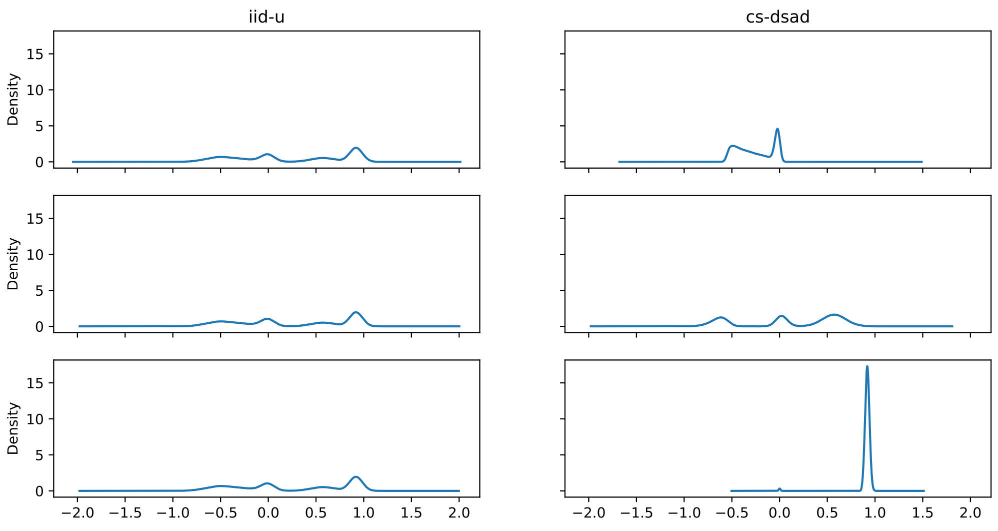


Figure 5.10: Comparison of the attribute '0' density from the expanded CURE dataset, for 'iid-u' and 'cs-dsad' data distributions with three partitions.

From the impact results for the DK-means algorithm in Table 5.6, we see a relatively high average impact for the 'cs-dsad' strategy. The attribute density distribution for this partitioning on the expanded CURE dataset is shown in Fig. 5.10. The points distribution in the two-dimensional space in Fig. 5.2 shows a linear layout of clusters, which provides class separation when applying the covariate shift strategy. *Shifting* data samples by attribute for datasets of low-dimensional space may lead to mixing categories of partitioning strategies, especially when classes are highly correlated with attributes. In this case, we partially mix the covariate shift and the prior probability shift strategies, as illustrated in the label distribution chart in Fig. 5.11. This test scenario suggests that mixed distributions, which are relatively common in the real world, influence more results than single corner cases.

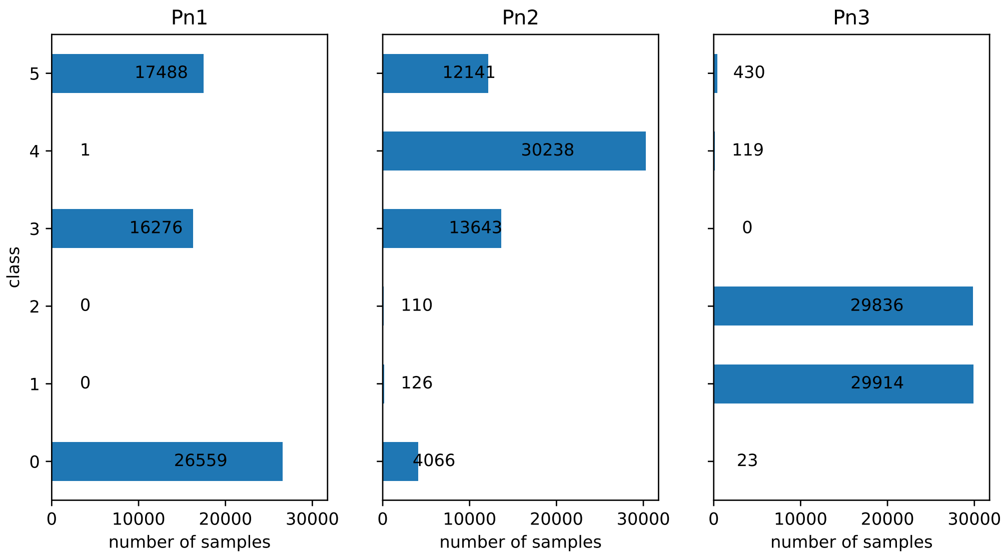


Figure 5.11: Data sample label distributions for the expanded CURE dataset, using the 'cs-dsad' strategy with three partitions.

To summarise our discussion of the approaches used here for distributed clustering, we recall that the k-means algorithm is known for producing different non-deterministic results, and adapting this to give a distributed version, propagates these differences into the subsequent processing stages. Although it can provide great results for fairly separated groups of data in a defined space, the mean results fluctuate enormously, making this algorithm unreliable in its basic form. The remedy, which can fix more than half of the uncertainties, is to use a density approach such as Opt-DKM. These results illustrate the first step in adapting an algorithm to work with unknown data partitioning. The next step is to improve the second half of uncertainties or drawbacks revealed during the current evaluation performed with more partitioning strategies, which is basically an extended version of the experiments reported in [49].

Table 5.6: Differences in the partitioning and density clustering quality results for partitioning strategies compared to a uniform distribution with the 'iid-u' strategy, where the heat map represents the five categories of partitioning impact.

	strategy	**3 nodes**				**6 nodes**				**12 nodes**			
algorithm	**dataset**	CURE-expanded	Diamond-expanded	Synthetic 7g	Synthetic 12g	CURE-expanded	Diamond-expanded	Synthetic 7g	Synthetic 12g	CURE-expanded	Diamond-expanded	Synthetic 7g	Synthetic 12g
d_dk-means	cs-dao	4.52%	2.93%	-1.49%	-.39%	4.81%	4.07%	5.94%	1.89%	4.35%	4.29%	-1.40%	.33%
	cs-dsad	6.90%	6.07%	4.19%	.27%	7.07%	8.28%	8.83%	1.71%	7.58%	6.42%	-3.35%	3.39%
	pps-s	5.10%	7.51%	6.66%	.47%	7.41%	8.77%	9.51%	2.75%	6.39%	7.87%	4.29%	8.37%
	pps-ab	4.15%	7.66%	3.77%	-1.12%	.24%	5.08%	6.78%	3.20%	4.27%	3.02%	-5.88%	5.21%
	pps-ms	2.39%	6.93%	4.10%	3.44%	6.26%	.84%	1.46%	2.21%	5.01%	2.03%	-6.74%	2.47%
	pps-mps	3.20%	6.80%	2.00%	.76%	5.70%	5.35%	9.06%	4.01%	6.49%	4.98%	-.38%	6.21%
	pps-mpa	4.82%	5.51%	7.89%	1.64%	6.24%	6.88%	5.85%	4.15%	1.55%	3.38%	-5.09%	3.05%
	cd-sba	2.89%	1.79%	-.26%	-1.30%	4.33%	3.43%	.86%	1.02%	-1.65%	3.35%	-3.09%	2.92%
	cs-cl	-.08%	2.11%	-1.48%	-3.73%	6.05%	.08%	2.11%	.90%	1.62%	-1.29%	-5.37%	.91%
	u-ul	2.59%	1.58%	.10%	-1.17%	4.54%	.07%	.70%	.09%	2.02%	.95%	-7.16%	.04%
d_opt-dkm	cs-dao	4.79%	.01%	1.74%	-1.42%	3.60%	-.91%	5.39%	-.19%	8.27%	-2.30%	3.86%	.71%
	cs-dsad	.49%	-.41%	4.29%	-8.45%	-.41%	-1.14%	-3.01%	-12.92%	.08%	-3.49%	1.82%	-15.08%
	pps-s	5.70%	-.44%	-3.64%	-8.45%	6.83%	-1.25%	-6.21%	-12.92%	6.98%	-4.15%	-5.36%	-15.08%
	pps-ab	.54%	1.30%	.65%	-3.79%	2.34%	-.14%	4.00%	7.20%	8.85%	-2.24%	8.34%	8.64%
	pps-ms	2.55%	-.33%	-3.63%	-8.45%	.24%	-.77%	-6.22%	-3.06%	-1.15%	-1.55%	2.59%	-.37%
	pps-mps	8.00%	.89%	-2.83%	-8.03%	1.37%	-1.18%	-4.51%	-12.92%	-1.78%	-4.02%	-4.63%	-15.08%
	pps-mpa	.52%	1.64%	1.33%	7.31%	.27%	.82%	4.20%	3.22%	2.01%	-2.25%	7.48%	1.74%
	cd-sba	5.14%	.23%	2.97%	.87%	6.39%	-.27%	1.58%	-4.86%	10.28%	-1.57%	4.82%	-6.63%
	cs-cl	1.13%	-.13%	4.35%	.64%	-.55%	-.03%	1.25%	.00%	.43%	.51%	.86%	-.43%
	u-ul	.19%	.48%	1.96%	.79%	-.29%	-.66%	.52%	-2.14%	.74%	2.97%	.93%	-.83%

Hierarchical ensemble From an analysis of the differences between the results from data distributed uniformly and with different partitioning strategies, we also find that a distributed implementation of an algorithm may not be entirely correct. When the difference is a relatively high negative value for most test cases, data distribution itself helps more to improve quality than the algorithm can achieve in a typical IID distribution. We would therefore define such an algorithm as unsuitable for unknown distributions. This situation arises for the distributed agglomerative BIRCH, where the difference in results compared to the 'iid-u' strategy is even higher than 14% in most cases, as shown in Table 5.7. This suggests that a hybrid combination of hierarchical and partitioning algorithms is a poor choice. On the other hand, it also reinforces the finding that the choice of algorithm for particular dataset characteristics is vital, as for the expanded Diamond dataset, this algorithm obtains perfect clustering results regardless of the strategy used. The only failure we observe occurs for more partitions in the 'cd-sba' strategy, where the hierarchical approach fails. The reason for this is that the attribute values start to become more noticeable on partitions and blend more with others and finally aggregate together. This is because other data samples are uniformly spread across all partitions. In contrast, drifted cluster samples remain distributed on a fixed number of partitions regardless of the worker number used in the experiment. An even higher impact could be achieved by simultaneously applying the partitioning strategy to more than one single label. A comparison of partitioning results for a uniform attribute density and the abovementioned distribution strategy is shown in Fig. 5.12.

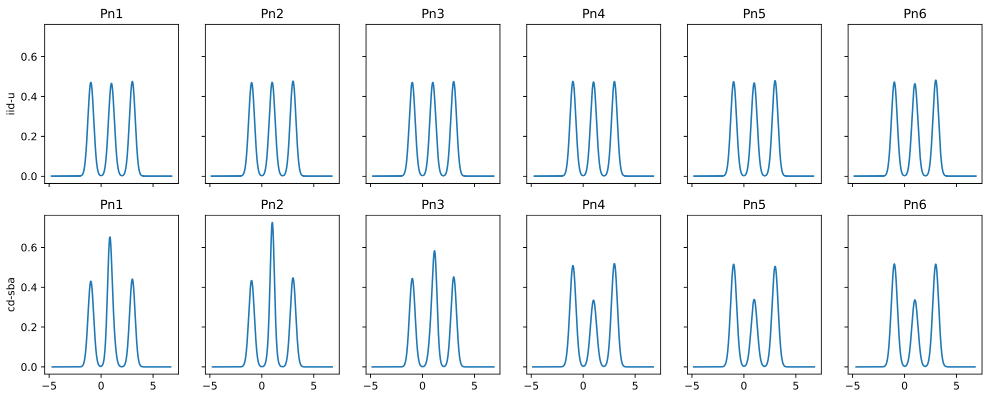


Figure 5.12: Comparison of the attribute '0' density from the extended Diamond dataset limited to data samples with class '4', for the 'iid-u' and 'cd-sba' data distributions with six partitions.

For the distributed agglomerative BIRCH algorithm, applying the 'cs-cl' strategy to the expanded CURE dataset provides worse results for clustering, which should be generally irrelevant for unsupervised method processing. This is because the initial data distribution matters when partitioning is performed. However, this partitioning is used in the global phase, and seems quite deterministic due to the small number of samples (local models). Hence, the worse results obtained here might be further grounds for suspecting the correctness of the algorithm in this distributed version.

Similarly to the DK-means pure partitioning algorithm discussed in the previous section, the ’cs-dsad’ strategy also substantially impacts the LCT algorithm with hierarchical enhancement at the global stage. This strategy also reveals that when more partitions are involved in clustering, the method of agglomeration of the found centroids fails to merge the local centroids into the correct number of final ones. Moreover, in the test case with 12 partitions of the Synthetic 7g dataset, all 84 of the local cluster centres found were merged into a single one, giving a meaningless result. Nevertheless, the overall poor results of the LCT algorithm are usually caused by finding an incorrect number of global clusters in the global stage of processing. This number of global clusters is typically too high even for ’iid-u’ partitioning. In addition, this implementation seems to be susceptible to the prior probability shift category, especially when any class representative is missing. The local partitioning approach then produces more inaccurate centroids, which misleads the hierarchical merging at the global stage. As a result, it merges wrong local cluster models, affecting the final quality results. In contrast, in the Opt-DKM algorithm, the imperfect centroids found at the local stage are correctly processed with a density-based approach.

Table 5.7: Differences in the hierarchical and partitioning clustering quality results for partitioning strategies compared to the uniform distribution with the 'iid-u' strategy, where the heat map shows the five categories of partitioning impact.

	strategy	**3 nodes**				**6 nodes**				**12 nodes**			
algorithm	**dataset**	CURE-expanded	Diamond-expanded	Synthetic 7g	Synthetic 12g	CURE-expanded	Diamond-expanded	Synthetic 7g	Synthetic 12g	CURE-expanded	Diamond-expanded	Synthetic 7g	Synthetic 12g
d_lct	cs-dao	-5.16%	-.85%	-12.83%	.17%	1.42%	.00%	-20.26%	-2.63%	1.67%	2.10%	.00%	-4.10%
	cs-dsad	.85%	1.58%	3.42%	3.42%	10.46%	12.56%	10.57%	8.91%	11.79%	18.04%	14.87%	10.97%
	pps-s	5.97%	2.38%	2.32%	6.91%	.22%	8.72%	8.30%	11.12%	11.22%	18.31%	14.56%	15.64%
	pps-ab	-2.47%	-4.43%	2.36%	-.33%	7.07%	3.53%	2.56%	1.48%	5.43%	6.73%	12.48%	3.08%
	pps-ms	3.09%	.72%	5.64%	4.65%	12.03%	8.60%	8.84%	4.49%	10.55%	11.36%	14.66%	5.31%
	pps-mps	1.25%	.32%	6.06%	4.51%	11.34%	12.52%	9.62%	11.06%	13.04%	17.62%	11.61%	14.54%
	pps-mpa	-4.16%	-8.66%	1.26%	-2.17%	2.17%	1.38%	-1.44%	-1.80%	2.65%	4.01%	14.80%	.63%
	cd-sba	.67%	-1.69%	-1.20%	1.42%	2.92%	.68%	-.78%	-1.20%	3.16%	2.48%	-.87%	.90%
	cs-cl	-3.89%	.15%	-.14%	.37%	-.73%	-.59%	-.02%	-2.22%	-1.43%	-.01%	-3.68%	-.06%
	u-ul	-.03%	-1.06%	2.03%	.99%	-1.86%	.41%	-1.06%	-.35%	-.24%	-.64%	1.09%	-.16%
d_dp-birch	cs-dao	.05%	.00%	-14.68%	-6.73%	14.89%	-.01%	-14.33%	-10.98%	21.32%	.00%	-13.77%	-.78%
	cs-dsad	1.02%	.00%	-14.88%	-1.64%	9.30%	-.01%	-.08%	-12.09%	10.72%	.00%	11.99%	-1.01%
	pps-s	-13.90%	.00%	-14.84%	-8.96%	-3.63%	-.02%	-14.88%	-27.90%	-17.65%	.00%	-14.85%	-17.45%
	pps-ab	-30.94%	.00%	-14.87%	-8.97%	-20.77%	-.02%	-.11%	-11.03%	-1.61%	.00%	-14.82%	-17.22%
	pps-ms	-30.50%	.00%	-14.83%	-8.99%	-3.70%	-.02%	-14.95%	-19.45%	-18.24%	.00%	-14.61%	-1.66%
	pps-mps	4.59%	.00%	-14.84%	-1.65%	-20.34%	.03%	-15.00%	-27.90%	-1.13%	.00%	-14.87%	.50%
	pps-mpa	1.12%	.00%	-14.76%	-17.43%	10.15%	-.01%	-14.81%	-12.08%	-2.03%	.01%	.00%	-17.41%
	cd-sba	-13.47%	.00%	.27%	-.16%	-3.16%	15.81%	-.09%	-10.47%	-1.50%	15.62%	-14.67%	-.47%
	cs-cl	4.96%	.00%	.01%	-.52%	10.18%	-.01%	-.04%	-10.93%	-1.65%	.01%	-14.52%	.02%
	u-ul	-8.69%	.00%	.00%	10.46%	-3.54%	-.01%	-.12%	-10.86%	-1.79%	.00%	-14.26%	-.13%

5.3.2 Processing time

In this section, we focus on the processing time statistics collected during the experiments. To avoid misleading results, we performed multiple executions even for deterministic algorithms to collect the average durations.

In a standard processing experiment, we can divide the processing time into several components, such as data loading, training, and evaluation. Evaluation takes time; for example, it might involve voting to create a prediction, possibly preceded by searching for the closest sample in the ensemble k-NN classifier sample, or assigning data samples to the global centres in partitioning clustering. In a distributed processing environment, we can identify additional time-related dimensions, such as the transfer and waiting times which occur in orchestration. These are present simultaneously on multiple computational nodes depending on the architecture, which for our evaluation includes multiple forms of local and global processing. The DDM-PS-Eval platform collects various statistics. For our evaluation of distributed algorithms with a central coordinator, we chose to compare the training process time as the sum of the maximum local and global processing times at each stage of the distributed algorithm pipeline. We excluded the transfer time, as our experiments were conducted on the local network.

The complete statistics for the datasets, algorithms and partitioning nodes are summarised in chart form in Figs. 7.4 and 7.5. When the processing time is relatively low in summary and lasts a few seconds, the slowdown and difference in the 12-node execution may result from the swapping of RAM memory and forced I/O operations rather than the global processing impact. To create an additional summary in the same way as for the quality results, we calculated an impact for each test case by comparing the training time to the same process duration with a uniform data distribution, using the formula $\frac{T_{strategy}}{T_{iid-u}-1}$. The collected and aggregated results for each average strategy value are presented in Table 5.8, using the same heat map as in the previous section. We treated the classification algorithms separately, as their time complexity is considerably different.

The impact on the processing for the clustering algorithms evaluated here is marginal for most non-IID strategies. Each evaluated clustering algorithm involves k-means partitioning, in which the time complexity is $O(n \cdot k \cdot i)$ (i.e. generally linear), where n is the number of samples, k is the number of groups, and i is the number of iterations. For Opt-DKM, the time complexity of the OPTICS algorithm is $O(N^2)$, but $N \ll n$ and depends on the number of partitions involved, where N is the number of local models at the global stage. The complexity of BIRCH can reach $O(N)$. We can therefore approximate all of these as linear for each dataset, and the speedup when using more computational nodes is noticeable only for Synthetic 12d, the largest dataset in terms of samples and numbers of dimensions. Statistics collected for other datasets regarding the processing time for each test case last less than five seconds, which does not allow us to draw meaningful conclusions. However, significant fluctuations are visible across test cases with different strategies, several of which consisted of substantially more samples to process.

Table 5.8: Differences in the processing time statistics for partitioning strategies compared to a uniform distribution with the 'iid-u' strategy, where the heat map represents the five categories of partitioning impact.

strategy \ method	Clustering	Classification: DN-SVM	Classification: Distributed Naive Bayes
cs-dao	16.12%	51.42%	12.83%
cs-dsad	3.60%	-8.80%	-2.06%
pps-s	7.44%	-0.61%	33.56%
pps-ab	4.04%	105.31%	27.70%
pps-ms	4.72%	69.50%	38.19%
pps-mps	6.65%	42.94%	37.44%
pps-mpa	4.55%	143.74%	32.53%
cd-sba	1.75%	138.41%	13.64%
cs-cl	0.69%	4.38%	-3.43%
u-ul	27.65%	318.70%	31.97%

The only significant differences in processing time are observed for the Synthetic 12g dataset using the 'cs-dao' and 'u-ul' strategies. The statistics obtained for each algorithm indicate that the solid high density of a similar data instances collated with a greater number of samples requires an algorithm to perform more iterations before it converges.

Unlike the processing of partitioning clustering, the data distribution significantly impacts classification tasks, especially when the time complexity is higher than linear. The standard SVM implementation time complexity is $O(N^3)$, which is vulnerable to the data size, whereas naive Bayes requires only a single pass to process the data. Hence, the time complexity for the second algorithm is $O(N \cdot d)$, where N is the number of data samples and d is the number of dimensions. In the execution of naive Bayes, the longer processing time is due to the linear increase in the data counted at some nodes compared to others, which affects the joining operation in the central node as it needs to wait for all the local results. However, the training time is so low that we cannot even observe a gain from the dispersal of calculations to multiple nodes.

A quick look at the results for another classifier indicates that an imbalance in the data samples does not negatively affect the quality but drastically affects the processing time. The SVM classifier is executed twice for the DN-SVN implementation, which increases the processing time when multiple support vectors are found at the local and global stages. A shorter processing time is found when the number of data samples is small and they are easily separable. The best illustration of a potentially unnoticed impact is the test case using the 'u-ul' strategy for the DN-SVN algorithm. Fig. 5.13 shows how unchanged quality results can go hand in hand with a drastic slowdown in processing. An extension to the processing time is also caused by the higher number of support vectors transferred, which increases the transfer load.

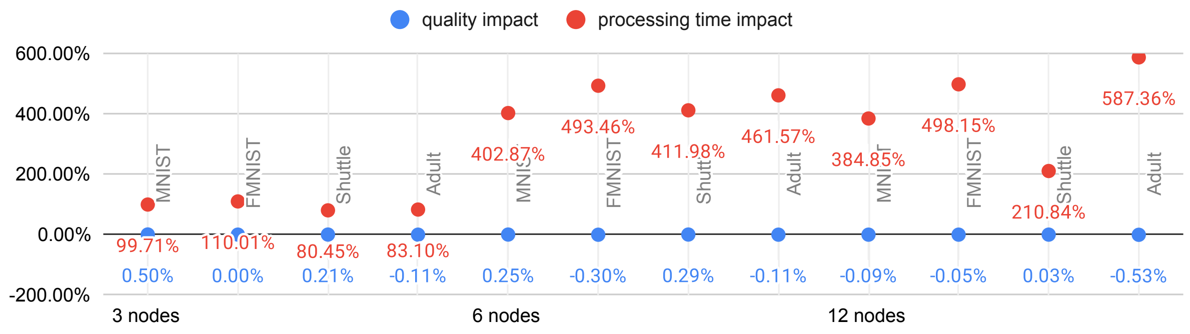


Figure 5.13: Correlation between impact on the quality and processing time for the DN-SVM algorithm, using the 'u-ul' strategy.

To summarise our analysis of the processing time, it is essential to note the hidden correlation between the number of data samples processed, the quality and the time required to process them. This is not a novel discovery, but is significant in specific data partitioning scenarios and has typically been overlooked in discussions of this issue, especially for underestimation of 'unbalancedness' partitioning in data during algorithm evaluation. We also observe various time impacts using other strategies, as the presence of class representatives influences the training process.

5.3.3 Transfer load

In the previous section, we discussed processing times and compared them to the quality results. Here, we briefly address the amounts of data transferred over the network depending on the data distribution. We collected the number of bytes transferred between local nodes and the central coordinator in the training process, including the costs due to the presence of data structures in serialised models if they occurred.

Data transfer naturally increases with the number of cooperating units; however, the load size can vary depending on the information collected locally, at least for local-to-global communication. In distributed algorithms with a central coordinator, data transfer typically starts from local nodes, which transfer their knowledge to the central node. This information may or may not be condensed, and is then resent to each cooperating unit. Hence, in this architecture, the transfer size for fixed-sized models increases with the number of collaborating units.

In our evaluation, the clustering algorithms based on the partitioning approach appeared to keep the constant size of the prepared models. The size of transferred bytes increased along with more nodes participating in training. The only fluctuations occurred when the density or hierarchy processing scheme caused the merging of clusters or transferring more small-sized cluster models. In the second situation, the transfer load increased slightly and generally gave only marginal differences within the processing of each algorithm. On the other hand, our evaluation showed that the communication overhead for the LCT algorithm is truly limited, as claimed by the authors, at least compared to the other two partitioning-based methods. Detailed results are shown in Fig. 7.7. However, this advantage was not translated into good-quality results.

As the classifiers used in this evaluation prepare models using a supervised approach, they may vary more from each other depending on the data distribution, especially when there are missing class representatives at several nodes. The most significant difference appears for the prior probability shift strategies in both classifiers, as shown by the results in Fig. 7.6. The size of the naive Bayes model depends only on the number of distinct labels, and hence is slightly larger for the 'cs-cl' strategy, in which artificial classes are present in the global scope. In the distributed version of the SVM algorithm examined here, the local models consist of locally found support vectors, and the overall transfer is therefore stable, regardless of the number of nodes involved. This is why a lower communication overhead is observed when more data samples are separated on distinct local nodes based on the class. The difference may be high in terms of absolute values, as illustrated in Fig. 5.14, where megabytes of data are transferred in the processing of the FMNIST dataset, for example. However, this lower transfer is typically seen with quality loss, as demonstrated in our quality impact analysis.

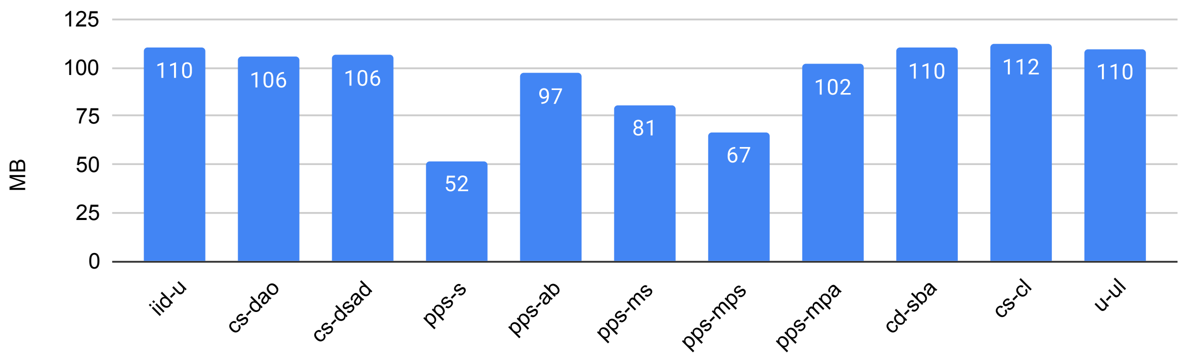


Figure 5.14: Data transfer results for the DN-SVM algorithm for various partitioning strategies during training on the FMNIST dataset, divided into three nodes.

In the same way as for non-distributed processing, more information can allow for the preparation of a better global model, or can lead to overfitting. In a distributed processing scheme, more information typically increases the transfer load. Nevertheless, how the model impacts quality results primarily depends on the specific processing of implemented method and the way of using carried information. From our experiments, we observed a linear increase in transfer size for the partitioning clustering methods, whereas the data transfer size for classification is more sensitive, as there is more variation between models depending on the training set. However, the relative difference in the transfer size for different data distributions seems to be substantial, and does not necessarily reduce the quality of the results.

The use of condensed models, which reduces data transfer, is more desirable as fewer data are shared outside the local site. An analysis of the transfer load may therefore reveal security issues, for example with the SVM implementation in which raw data are transferred between computational nodes. Moreover, although they are not included in this work, additional statistics on transfer durations obtained from our evaluation of the DDM-PS-Eval platform show that the model size corresponds to the extended communication time, even when training is fast. A side effect such as this arising from the size of the model may be crucial for collaborative algorithms, as it significantly increases delays and must be controlled in the design of a distributed algorithm.

5.4 Summary of negative impact

In this section, we present a summary of multiple experiments. To confirm the relationship between partitioning strategy and algorithm processing, we collected quality and time impact data and compared them to the results from a uniform distribution (IID) by averaging the combined results obtained from test cases for each main non-IID category. However, these values may be underestimated, as some correct results indicate a low impact due to incorrect strategy parameterisation rather than algorithm resistance, which understates the average values in the presented summary for the categories. This is why the use of a wide range of distribution strategies is required for the proper evaluation of algorithms.

Depending on the behaviour of the algorithm, the results may be better or worse, or may not change at all. For example, in the case of distributed versions of density-based algorithms, data separation is typically an obstacle in subsequent phases of global aggregation. For this reason, it is essential to validate algorithms using multiple data partitioning strategies rather than a single uniform one. In addition, we need to remember that the impact of non-IID data partitioning on the quality of the algorithms also depends on the characteristics of the data themselves, without partitioning. The detailed results in the attachment show that the distributions of the results for the same algorithm differ between the datasets; for instance, this is clearly visible for the distributed agglomerative BIRCH algorithm.

Although it is not easy to visualise multiple results, it is clear that specific data distributions strongly affect the processing of different algorithms. It is therefore possible to determine which data partitioning strategy will negatively affect the algorithm processing in terms of final quality results and execution statistics depending on the type of algorithm. Unfortunately, a fully confident and correct classification is impossible for hybrid algorithms, but we can infer the scale of the impact based on the influence of the algorithm on a crucial aspect of the processing. However, we can undoubtedly identify which groups of algorithms to avoid when processing data with a particular distribution if this is known or suspected. The complete classification requires yet many more different algorithms evaluation for each group. After analysing the theoretical basis of operation for the implementations evaluated here, we can draw initial conclusions on the scale of the impact for particular types of algorithm used at different stages of processing. A summary of the maximum average impact on quality and processing time in each category is presented in Table 5.9. These average results do not require further discussion, as they are straightforward and were analysed individually in an earlier section.

Table 5.9: Comparison of the impact of quality and processing time on algorithms executed with different data distributions to the results for a uniform data distribution (the heat map follows the scheme in the tables above: L, M, and H indicate low ($< 5\%$), medium ($< 10\%$), high ($\geq 10\%$) impact, respectively; 0 indicates no noticeable impact ($[-2\%, 2\%]$); and P describes a positive impact where the results are better than those obtained for a uniform data distribution).

	Quality impact						**Processing time impact**					
Local algorithm type	**SVM**	**Probabilistic**	**Partitioning**	**Partitioning**	**Partitioning**	**Hierarchical**	**SVM**	**Probabilistic**	**Partitioning**	**Partitioning**	**Partitioning**	**Hierarchical**
Global algorithm type			**Partitioning**	**Density**	**Hierarchical**	**Partitioning**			**Partitioning**	**Density**	**Hierarchical**	**Partitioning**
Main non-IID category												
Covariate shift	0	0	L	0	M	0	H	*H*	H	H	H	H
Prior probability shift	H	0	M	L	M	P	H	*H*	M	M	M	M
Concept drift	L	0	0	0	0	0	H	*H*	L	0	L	0
Concept shift	H	M	*n/a*	*n/a*	*n/a*	*n/a*	L	*P*	*n/a*	*n/a*	*n/a*	*n/a*
Unbalancedness	0	0	0	0	0	P	H	*H*	H	H	H	H

The summarised scale of the gathered negative quality impact illustrated in Table 5.9 maps to the suggested minimal test suite coverage depicted in Fig. 4.3 that suppose to be performed during algorithm evaluation. It is especially accurate for clustering methods confirming validity of this minimal evaluation set. Table 5.9 presents aggregated results, but when we look closer at the particular result values obtained by applying nested strategies, we can potentially extract the next level of subcategories in our taxonomy. For instance, unlike the clustering algorithms, the tested classifier suffers significantly more from the 'cs-dao' distribution strategy than from 'cs-dsad' while both belong to the 'covariate shift' category.

With regard to transfer size, it is impossible to define the impact of data distribution for any group of algorithms, as the models are too diverse. The model is the core of the algorithm and therefore dedicated to the method. However, the impact on transfer using simple techniques is usually marginal, although it can play a more prominent role more complex methods, and we therefore need to monitor their sizes in a benchmarking comparison.

Chapter 6

Conclusion

The main goal of this work was to analyse, standardise and enhance the evaluation process for distributed clustering and classification algorithms with uneven data distribution. We introduced new data partitioning strategies to simulate different non-IID data distributions to achieve this goal. For the sake of completeness of the standardised evaluation process, we developed a novel evaluation platform, which was designed to perform a comparison of multidimensional experiments with different non-IID data distributions. We demonstrated a negative impact on the execution of distributed algorithms working with a local independent dataset in terms of various aspects of processing, such as the final quality, slowdown in performance and network transfer increase. Moreover, we proposed a hybrid distributed clustering algorithm that overcomes most quality failures originating from poor local initialisation for more than half of the test cases compared to the primary reference approach. In the experimental section, we evaluated two types of classifier to show the strong impact or neutrality of the data distribution influence on the algorithm processing. We examined several methods based on the common core partitioning approach to clustering evaluation. A brief analysis of the results obtained from our evaluation of partitioning strategies using different types of distributed algorithms allowed us to identify a potential scale for the impact of the data distribution on the results.

Of the main goals of this work, we can see that the secondary objectives have been accomplished. We extended and standardised a non-IID taxonomy. We also extracted and confirmed the validity of a minimal test suite coverage for the correct algorithm evaluation. To examine the data partitioning strategies, we prepared a set of generic parameterised test suites that are ready to use on different datasets, which can reveal potential drawbacks in newly designed algorithms.

We presented a hybrid concept that combines two different types of clustering algorithms for single distribution processing: k-means is used as a base partitioning algorithm, and a density-based OPTICS algorithm is used to build global clustering models. The input parameters for OPTICS are determined automatically, based on our local clustering models, and no user input is required. Our method also fixes the issue of poor local initialisation for k-means, thus eliminating the influence of the bad start problem, so that even random initialisation at the local stage does not reduce the quality of the results. Parameterisation allows us to improve the global phase by calculating the real number of clusters for the entire distributed dataset. The proposed density approach improved the local k-means initialisation accuracy in the global clustering phase, which led to improvements in the overall average quality compared with local clustering and hybrid methods utilising a hierarchical approach.

As part of this work, we have introduced a novel platform for evaluating DDM algorithms that considered the impact of the data distribution. The platform, implemented from scratch with PoC elements, is easy to use and extend by user-defined, pluggable implementations of machine learning algorithms using the provided framework. We have also addressed the often neglected impact of data partitioning on the results by integrating strategy mechanisms directly into our platform. The proposed evaluation tool can be readily used in practice to compare new DDM algorithms in terms of speed, network load, and the quality of the results using any dataset.

The proposed partitioning methods have been evaluated for distributed clustering and classification algorithms using several datasets and various algorithms. Each data partitioning strategy has been implemented and integrated with the DDM-PS-Eval platform. In conclusion, this study has shown that data partitioning has a significant impact on the results provided by distributed algorithms. This work paves the way for better validation of algorithms, in order to allow for the design of algorithms that are agnostic in terms of data distribution.

6.1 Limitations

We aimed to determine a generalised impact by conducting a wide range of test cases using multiple partitions, datasets and algorithms; however, one of the results of our experimental evaluation analysis was that there are multiple variables that affect strategy evaluation. For different ensemble methods, the precise impact is difficult to determine, as algorithms may use multiple hyperparameters to deal with different situations. Hence, a deeper insight and analysis of more realistic datasets and dedicated algorithm types for practical purposes is still required, in order to reveal more specific implications.

In most cases, the impact on the data transfer seems straightforward for the simple models built by the evaluated algorithms, with a central communication architecture. More experiments with other distribution architectures, such as P2P networks with more sophisticated algorithms, are required to discover more consequences in terms of the transfer load.

The proposed hybrid clustering algorithm overcomes the obstacles arising from initialisation with certain data distributions. However, the introduction of more detailed partitioning strategies and the collection of evaluation statistics using the implemented platform revealed some shortcomings in the processing quality and duration that need to be addressed.

One possible drawback of the proposed evaluation platform is the technology selected, since in recent years, we have observed significant increases in the popularity of Python and the numbers of works and algorithm implementations written in this language. Although the JVM-based language is still popular, modern PoC programs typically rely on Python, which makes it harder to evaluate the latest works. Further work would be required in terms of porting or wrapping the implementations to compare execution on the platform, which would introduce some additional costs for statistics. However, if the platform could handle the algorithms written in Python, we could attach more recent implementations in evaluation, even the FL ones.

6.2 Future research

Besides the shortcomings described in the previous section, the research presented here opens several possible avenues for further study. We can divide these into three categories: the design of distributed and robust algorithms, platform development in the areas of automatisation and UI, and further investigation of the presence of non-IID data in more practical applications.

Regarding the DDM-PS-Eval platform, there is a never-ending list of tasks for further development, including optimisation and new features for other types of data and algorithms. Current process of preparation experiments requires some additional semi-automatic setup, such as manual choosing an attribute or label to process as a strategy parameter. However, this choice can be successfully implemented within the partitioning strategies to work automatically.

We have evaluated only a small selection of the existing types of classification and clustering methods and implementations. An incremental, systematic study of the different data mining methods is still required to obtain a better view of the side effects of data partitioning. Nevertheless, this is not an easy task due to the diversity in the approaches, algorithm architectures, and data types used. Further research on different data partitioning schemes, such as for textual datasets, is necessary to extend the taxonomy introduced here and the possibilities for evaluation.

With regard to partitioning methods, future work will involve extending the proposed methods to produce more realistic distributions of non-IID data by mixing multiple partitioning strategies based on attributes, labels, and quantity. Several open issues need to be analysed and addressed, such as the aforementioned mixture of strategies and the ability to deal with a large number of partitions for data with few labels. Further investigation of strategy parameterisation is also required. As discussed in our analysis, incorrect parameterisation of a single partitioning strategy may produce data partitioning that mix different non-IID categories. Nevertheless, the methods proposed here represent only a few possibilities for the data distributions, and many more are possible.

Bibliography

[1] Aisha Abdallah, Mohd Aizaini Maarof, and Anazida Zainal. Fraud detection system: A survey. *Journal of Network and Computer Applications*, 68:90–113, 2016.

[2] Elke Achtert, Hans-Peter Kriegel, and Arthur Zimek. Elki: a software system for evaluation of subspace clustering algorithms. In *International Conference on Scientific and Statistical Database Management*, pages 580–585. Springer, 2008.

[3] Mouhib Alnoukari. From business intelligence to big data: The power of analytics. In *Research Anthology on Big Data Analytics, Architectures, and Applications*, pages 823–841. IGI Global, 2022.

[4] Mihael Ankerst, Markus M Breunig, Hans-Peter Kriegel, and Jörg Sander. Optics: Ordering points to identify the clustering structure. *ACM Sigmod record*, 28(2):49–60, 1999.

[5] Lamine M Aouad, Nhien-An Le-Khac, and Tahar M Kechadi. Lightweight clustering technique for distributed data mining applications. In *Industrial conference on data mining*, pages 120–134. Springer, 2007.

[6] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. Technical report, Stanford, 2006.

[7] Árpád Berta, István Hegedűs, and Róbert Ormándi. Lightning fast asynchronous distributed k-means clustering. 2014.

[8] Albert Bifet, Geoff Holmes, Bernhard Pfahringer, Philipp Kranen, Hardy Kremer, Timm Jansen, and Thomas Seidl. Moa: Massive online analysis, a framework for stream classification and clustering. In *Proceedings of the first workshop on applications of pattern analysis*, pages 44–50. PMLR, 2010.

[9] Christoph Boden, Alexander Alexandrov, Andreas Kunft, Tilmann Rabl, and Volker Markl. Peel: A framework for benchmarking distributed systems and algorithms. In *Technology Conference on Performance Evaluation and Benchmarking*, pages 9–24. Springer, 2017.

[10] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečnỳ, Stefano Mazzocchi, Brendan McMahan, et al. Towards federated learning at scale: System design. *Proceedings of Machine Learning and Systems*, 1:374–388, 2019.

[11] Houda Bouraqqadi, Ayoub Berrag, Mohamed Mhaouach, Afaf Bouhoute, Khalid Fardousse, and Ismail Berrada. Pyfed: extending pysyft with n-iid federated learning benchmark. In *The 34th Canadian Conference on Artificial Intelligence*, 2021.

[12] Sebastian Caldas, Sai Meher Karthik Duddu, Peter Wu, Tian Li, Jakub Konečnỳ, H Brendan McMahan, Virginia Smith, and Ameet Talwalkar. Leaf: A benchmark for federated settings. *arXiv preprint arXiv:1812.01097*, 2018.

[13] Min Chen, Shiwen Mao, and Yunhao Liu. Big data: A survey. *Mobile networks and applications*, 19(2):171–209, 2014.

[14] Seung-Seok Choi, Sung-Hyuk Cha, and Charles C Tappert. A survey of binary similarity and distance measures. *Journal of systemics, cybernetics and informatics*, 8(1):43–48, 2010.

[15] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre Van Schaik. Emnist: Extending mnist to handwritten letters. In *2017 international joint conference on neural networks (IJCNN)*, pages 2921–2926. IEEE, 2017.

[16] Souptik Datta, Chris Giannella, and Hillol Kargupta. K-means clustering over a large, dynamic network. In *Proceedings of the 2006 SIAM international conference on data mining*, pages 153–164. SIAM, 2006.

[17] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[18] Ciprian Dobre and Fatos Xhafa. Intelligent services for big data science. *Future generation computer systems*, 37:267–281, 2014.

[19] Adil Fahad, Najlaa Alshatri, Zahir Tari, Abdullah Alamri, Ibrahim Khalil, Albert Y Zomaya, Sebti Foufou, and Abdelaziz Bouras. A survey of clustering algorithms for big data: Taxonomy and empirical analysis. *IEEE transactions on emerging topics in computing*, 2(3):267–279, 2014.

[20] Alec Go, Richa Bhayani, and Lei Huang. Twitter sentiment classification using distant supervision. *CS224N project report, Stanford*, 1(12):2009, 2009.

[21] Sorin M Grigorescu. Generative one-shot learning (gol): A semi-parametric approach to one-shot learning in autonomous vision. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7127–7134. IEEE, 2018.

[22] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. Cure: An efficient clustering algorithm for large databases. *ACM Sigmod record*, 27(2):73–84, 1998.

[23] John A Hartigan and Manchek A Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the royal statistical society. series c (applied statistics)*, 28(1):100–108, 1979.

[24] Yue He, Zheyan Shen, and Peng Cui. Towards non-iid image classification: A dataset and baselines. *Pattern Recognition*, 110:107383, 2021.

[25] Mohammad Hossin and Md Nasir Sulaiman. A review on evaluation metrics for data classification evaluations. *International journal of data mining & knowledge management process*, 5(2):1, 2015.

[26] Kevin Hsieh, Amar Phanishayee, Onur Mutlu, and Phillip Gibbons. The non-iid data quagmire of decentralized machine learning. In *International Conference on Machine Learning*, pages 4387–4398. PMLR, 2020.

[27] Tzu-Ming Harry Hsu, Hang Qi, and Matthew Brown. Measuring the effects of non-identical data distribution for federated visual classification. *arXiv preprint arXiv:1909.06335*, 2019.

[28] Sixu Hu, Yuan Li, Xu Liu, Qinbin Li, Zhaomin Wu, and Bingsheng He. The oarf benchmark suite: Characterization and implications for federated learning systems. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 13(4):1–32, 2022.

[29] Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of classification*, 2(1):193–218, 1985.

[30] László A Jeni, Jeffrey F Cohn, and Fernando De La Torre. Facing imbalanced data–recommendations for the use of performance metrics. In *2013 Humaine association conference on affective computing and intelligent interaction*, pages 245–251. IEEE, 2013.

[31] Genlin Ji and Xiaohan Ling. Ensemble learning based distributed clustering. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 312–321. Springer, 2007.

[32] Ruoming Jin, Anjan Goswami, and Gagan Agrawal. Fast and exact out-of-core and distributed k-means clustering. *Knowledge and Information Systems*, 10(1):17–40, 2006.

[33] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning*, 14(1–2):1–210, 2021.

[34] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank J Reddi, Sebastian U Stich, and Ananda Theertha Suresh. Scaffold: Stochastic controlled averaging for on-device federated learning. 2019.

[35] Burhanullah Khattak, Aurangzeb Khan, Khairullah Khan, Wahab Khan, Muhammad Kamran, and Muhammad Fahad. Empirical analysis of recent advances, characteristics and challenges of big data. *EAI Endorsed Transactions on Scalable Information Systems*, 6(23), 2019.

[36] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[37] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[38] Marzena Kryszkiewicz and Piotr Lasek. Ti-dbscan: Clustering with dbscan by means of the triangle inequality. In *International Conference on Rough Sets and Current Trends in Computing*, pages 60–69. Springer, 2010.

[39] Wiktor Kuśmirek, Agnieszka Szmurło, Marek Wiewiórka, Robert Nowak, and Tomasz Gambin. Comparison of knn and k-means optimization methods of reference set selection for improved cnv callers performance. *BMC bioinformatics*, 20(1):1–10, 2019.

[40] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[41] Ang Li, Jingwei Sun, Binghui Wang, Lin Duan, Sicheng Li, Yiran Chen, and Hai Li. Lotteryfl: Personalized and communication-efficient federated learning with lottery ticket hypothesis on non-iid datasets. *arXiv preprint arXiv:2008.03371*, 2020.

[42] Qinbin Li, Yiqun Diao, Quan Chen, and Bingsheng He. Federated learning on non-iid data silos: An experimental study. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, pages 965–978. IEEE, 2022.

[43] Qinbin Li, Zeyi Wen, Zhaomin Wu, Sixu Hu, Naibo Wang, Yuan Li, Xu Liu, and Bingsheng He. A survey on federated learning systems: vision, hype and reality for data privacy and protection. *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[44] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. *Proceedings of Machine Learning and Systems*, 2:429–450, 2020.

[45] Xavier Limón, Alejandro Guerra-Hernández, Nicandro Cruz-Ramírez, and Francisco Grimaldo. Modeling and implementing distributed data mining strategies in jaca-ddm. *Knowledge and Information Systems*, 60(1):99–143, 2019.

[46] Lifeng Liu, Fengda Zhang, Jun Xiao, and Chao Wu. Evaluation framework for large-scale federated learning. *arXiv preprint arXiv:2003.01575*, 2020.

[47] Yanchi Liu, Zhongmou Li, Hui Xiong, Xuedong Gao, and Junjie Wu. Understanding of internal clustering validation measures. In *2010 IEEE international conference on data mining*, pages 911–916. IEEE, 2010.

[48] Jiahuan Luo, Xueyang Wu, Yun Luo, Anbu Huang, Yunfeng Huang, Yang Liu, and Qiang Yang. Real-world image datasets for federated learning. *arXiv preprint arXiv:1910.11089*, 2019.

[49] Mikołaj Markiewicz and Jakub Koperwas. Hybrid partitioning-density algorithm for k-means clustering of distributed data utilizing optics. *International Journal of Data Warehousing and Mining (IJDWM)*, 15(4):1–20, 2019.

[50] Mikołaj Markiewicz and Jakub Koperwas. Evaluation platform for ddm algorithms with the usage of non-uniform data distribution strategies. *International Journal of Information Technologies and Systems Approach (IJITSA)*, 15(1):1–23, 2022.

[51] Mikołaj Markiewicz. and Jakub Koperwas. Data partitioning strategies for simulating non-iid data distributions in the ddm-ps-eval evaluation platform. In *Proceedings of the 17th International Conference on Software Technologies - ICSOFT,*, pages 307–318. INSTICC, SciTePress, 2022.

[52] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.

[53] Dirk Merkel et al. Docker: lightweight linux containers for consistent development and deployment. *Linux j*, 239(2):2, 2014.

[54] Behroz Mirza, Tahir Q Syed, Behraj Khan, and Yameen Malik. Potential deep learning solutions to persistent and emerging big data challenges—a practitioners' cookbook. *ACM Computing Surveys (CSUR)*, 54(1):1–39, 2021.

[55] Antonio Mucherino, Petraq J Papajorgji, Panos M Pardalos, Antonio Mucherino, Petraq J Papajorgji, and Panos M Pardalos. K-nearest neighbor classification. *Data mining in agriculture*, pages 83–106, 2009.

[56] Angel Navia-Vázquez, D Gutierrez-Gonzalez, Emilio Parrado-Hernández, and JJ Navarro-Abellan. Distributed support vector machines. *IEEE Transactions on Neural Networks*, 17(4):1091, 2006.

[57] Silvia Nittel, Kelvin T Leung, and Amy Braverman. Scaling clustering algorithms for massive data sets using data streams. In *ICDE*, volume 4, page 830, 2004.

[58] Gabriele Oliva, Roberto Setola, and Christoforos N Hadjicostis. Distributed k-means algorithm. *arXiv preprint arXiv:1312.4176*, 2013.

[59] Lior Rokach and Oded Maimon. Clustering methods. In *Data mining and knowledge discovery handbook*, pages 321–352. Springer, 2005.

[60] Simone Romano, Nguyen Xuan Vinh, James Bailey, and Karin Verspoor. Adjusting for chance clustering comparison measures. *The Journal of Machine Learning Research*, 17(1):4635–4666, 2016.

[61] Stan Salvador and Philip Chan. Determining the number of clusters/segments in hierarchical clustering/segmentation algorithms. In *16th IEEE international conference on tools with artificial intelligence*, pages 576–584. IEEE, 2004.

[62] Ohad Shamir, Nati Srebro, and Tong Zhang. Communication-efficient distributed optimization using an approximate newton-type method. In *International conference on machine learning*, pages 1000–1008. PMLR, 2014.

[63] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*, pages 1–10. Ieee, 2010.

[64] Toon Van Craenendonck and Hendrik Blockeel. Using internal validity measures to compare clustering algorithms. *Benelearn 2015 Poster presentations (online)*, pages 1–8, 2015.

[65] Geoffrey I Webb, Eamonn Keogh, and Risto Miikkulainen. Naïve bayes. *Encyclopedia of machine learning*, 15:713–714, 2010.

[66] Jian Wei, Kai Chen, Yi Zhou, Qu Zhou, and Jianhua He. Benchmarking of distributed computing engines spark and graphlab for big data analytics. In *2016 IEEE Second International Conference on Big Data Computing Service and Applications (BigDataService)*, pages 10–13. IEEE, 2016.

[67] Ian H Witten and Eibe Frank. Data mining: practical machine learning tools and techniques with java implementations. *Acm Sigmod Record*, 31(1):76–77, 2002.

[68] Shanshan Wu, Tian Li, Zachary Charles, Yu Xiao, Ziyu Liu, Zheng Xu, and Virginia Smith. Motley: Benchmarking heterogeneity and personalization in federated learning. *arXiv preprint arXiv:2206.09262*, 2022.

[69] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[70] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauly, Michael J Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A {Fault-Tolerant} abstraction for {In-Memory} cluster computing. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 15–28, 2012.

[71] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: A new data clustering algorithm and its applications. *Data mining and knowledge discovery*, 1(2):141–182, 1997.

[72] Yudian Zheng, Reynold Cheng, Silviu Maniu, and Luyi Mo. On optimality of jury selection in crowdsourcing. In *EDBT*, pages 193–204, 2015.

[73] Hangyu Zhu, Jinjin Xu, Shiqing Liu, and Yaochu Jin. Federated learning on non-iid data: A survey. *Neurocomputing*, 465:371–390, 2021.

Chapter 7

Attachments

7.1 DDM-PS-Eval technical details

7.1.1 Communication

The central point of the system is the coordinator application. It forms the entry point to the algorithm execution and environment setup for the system; moreover, is implemented as a web application and exposes an API for interaction. In general, the coordinator application can be seen as a test suite container and a communication proxy with workers. Its role is to set up a test environment, control the health checks of the instances, and schedule multiple executions to verify the statistical significance of results produced by non-deterministic algorithms. All the data provided by the user and the collected results are stored in the integrated storage component. Execution results are archived on shutdown, while algorithms are conditionally cleared. The application responsible for communication with the coordinator application and communication between other nodes in the environment is the node agent. This physically controls the execution of the driver program for experiments with a given configuration of the algorithm, data, and other execution parameters. Each agent reports its execution status and provides execution logs and results with statistics.

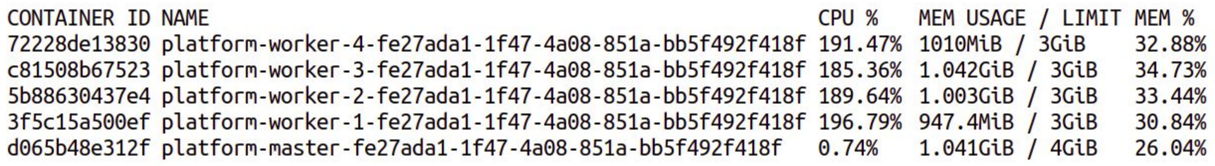
The agent application is currently implemented in Java version 8, for compatibility with Spark 2.4.5, which is used in the proof of concept (PoC). At the current stage of implementation, the driver program is written as a Spark application packaged in a .jar file, and runs on a cluster built on Spark workers installed on the worker nodes. The issues related to communication when using Spark are discussed in a later section. The driver is responsible for the execution of the algorithm, gathering processing time and transfer statistics, and orchestration of the execution pipeline. The development script is a local script written in Python that executes sequences of API calls wrapped in functions that set up, reload resources, execute, or provide execution information, such as logs, statuses, results, etc.

Current communication drawbacks

Our current solution is based on collaborating Docker containers, consisting of the node agent application and a Spark node. A Spark worker is started on each worker node, while the Spark master is on the central node. This hidden Spark environment in the Docker container has been used for PoC purposes in order to demonstrate the system's capabilities, but it can easily be removed. We have used many of the 'out of the box' features provided by Spark, such as worker communication and code transfer; however, this also imposes several limitations. The Spark environment builds strongly connected clusters and performs multiple operations that are hard or even impossible to control, meaning that work must be done within the same network. Furthermore, the driver program can only pass hints to the Spark execution scheduler about which nodes to use by providing preferable data locations, but cannot force this use in any way. Nevertheless, it is possible to set up sufficiently strict configuration parameters and hints to make it work as expected, and we have done this mostly by setting low communication timeouts, ports, and preferred locations. To ensure confidence in our current solution, verification is always done to check that every node with its local data in the instance has participated in execution; otherwise, the execution fails. As we are satisfied that the system worked as expected, we plan to remove the hidden Spark components in favour of explicit execution by the dedicated protocol of the node agents on the worker nodes.

7.1.2 Environment setup

Distributed systems need to work in an authentic distributed manner, although local testing can be beneficial. Our platform was therefore equipped with this possibility, and the coordinator application can operate in two profiles: manual or local setup. In the first case, the user provides an existing infrastructure setup consisting of nodes with agents installed. This setup contains some environmental information but mostly addresses, as shown in Listing 7.1. The second profile allows us to create and destroy local Docker images with workers and to apply specific limitations to containers, as depicted in Fig. 7.1. Regardless of the configuration, multiple setups known as 'instances' can exist simultaneously. Each instance consists of a specific number of workers described by a network address, and the open ports used, the number of cores, the available memory, and the disk size. One additional worker always acts as a DDM central coordinator.



```
CONTAINER ID NAME                                               CPU %    MEM USAGE / LIMIT MEM %
72228de13830 platform-worker-4-fe27ada1-1f47-4a08-851a-bb5f492f418f 191.47% 1010MiB / 3GiB    32.88%
c81508b67523 platform-worker-3-fe27ada1-1f47-4a08-851a-bb5f492f418f 185.36% 1.042GiB / 3GiB   34.73%
5b88630437e4 platform-worker-2-fe27ada1-1f47-4a08-851a-bb5f492f418f 189.64% 1.003GiB / 3GiB   33.44%
3f5c15a500ef platform-worker-1-fe27ada1-1f47-4a08-851a-bb5f492f418f 196.79% 947.4MiB / 3GiB   30.84%
d065b48e312f platform-master-fe27ada1-1f47-4a08-851a-bb5f492f418f   0.74%   1.041GiB / 4GiB   26.04%
```

Figure 7.1: Docker container limits when running an experiment, obtained with the Docker stats command.

Listing 7.1: Sample manual setup for platform consisting of agents located in the environment and open ports used in communication.

```
{ "nodes": [ {
     "address": "192.168.1.23",
     "agentPort": "10050",
     "cpu": 4,
     "memoryInGb": 2,
     "name": "new-master",
     "port": "10000",
     "type": "master",
     "uiPort": "10030"
    }, {
     "address": "192.168.2.58",
     "agentPort": "10051",
     "cpu": 4,
     "memoryInGb": 2,
     "name": "new-worker-1",
     "port": "10001",
     "type": "worker",
     "uiPort": "10031"
    }, {
     "address": "192.168.2.13",
     "agentPort": "10052",
     "cpu": 4,
     "memoryInGb": 2,
     "name": "new-worker-2",
     "port": "10002",
     "type": "worker",
     "uiPort": "10032"
    } ]
```

7.2 Detailed quality results

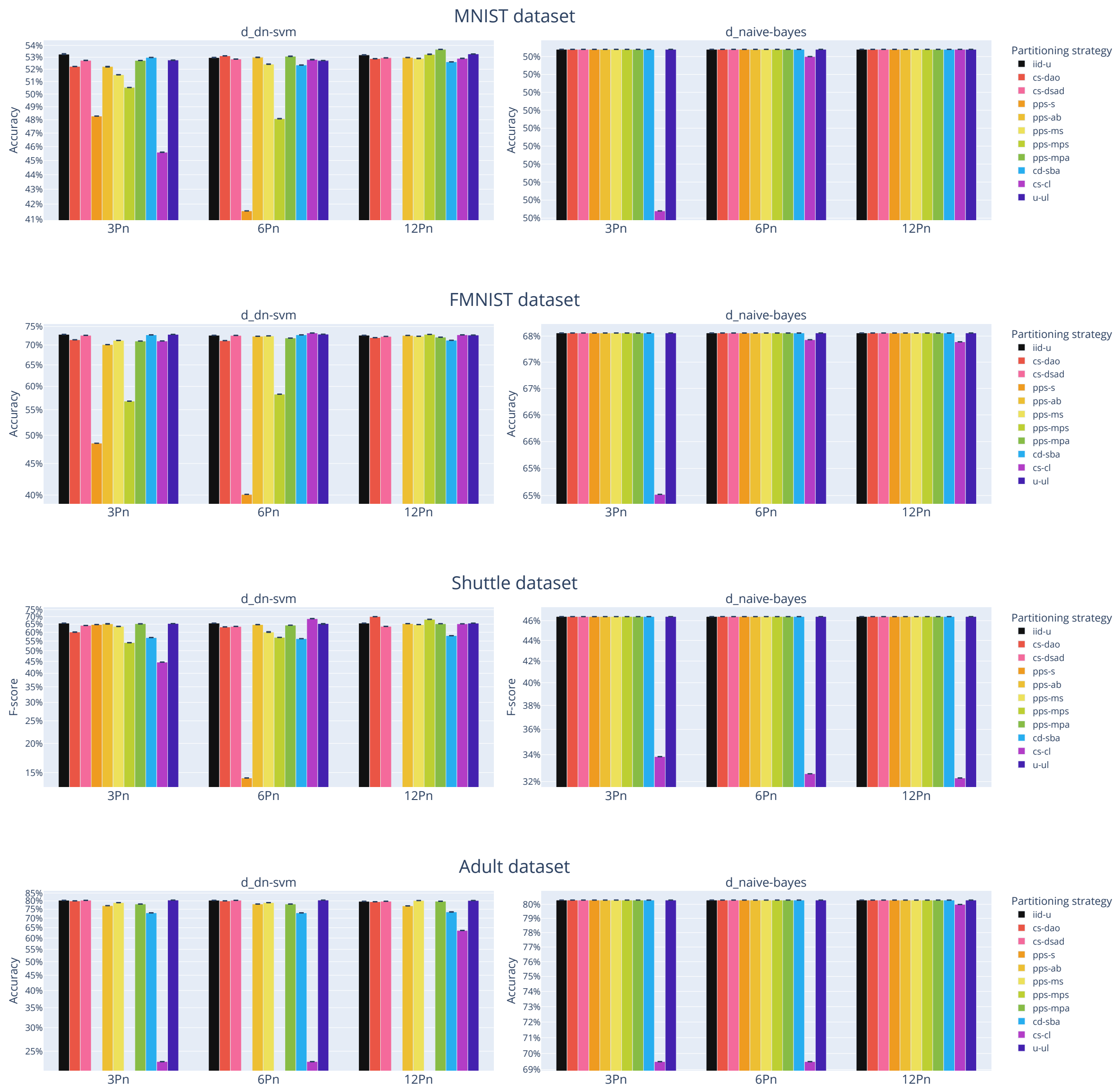


Figure 7.2: Detailed quality results for classification evaluation.

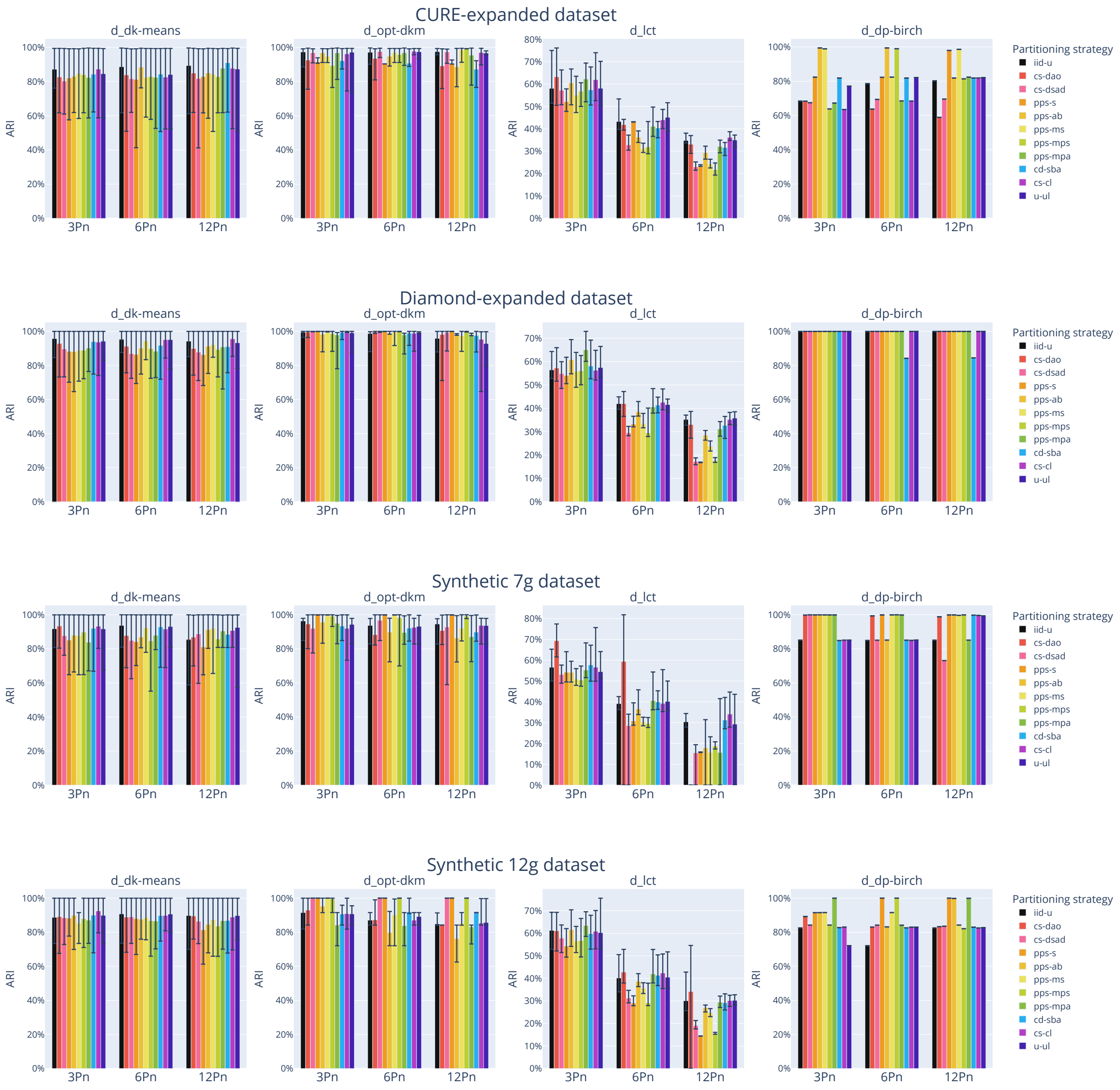


Figure 7.3: Detailed quality results for clustering evaluation.

7.3 Detailed processing time statistics

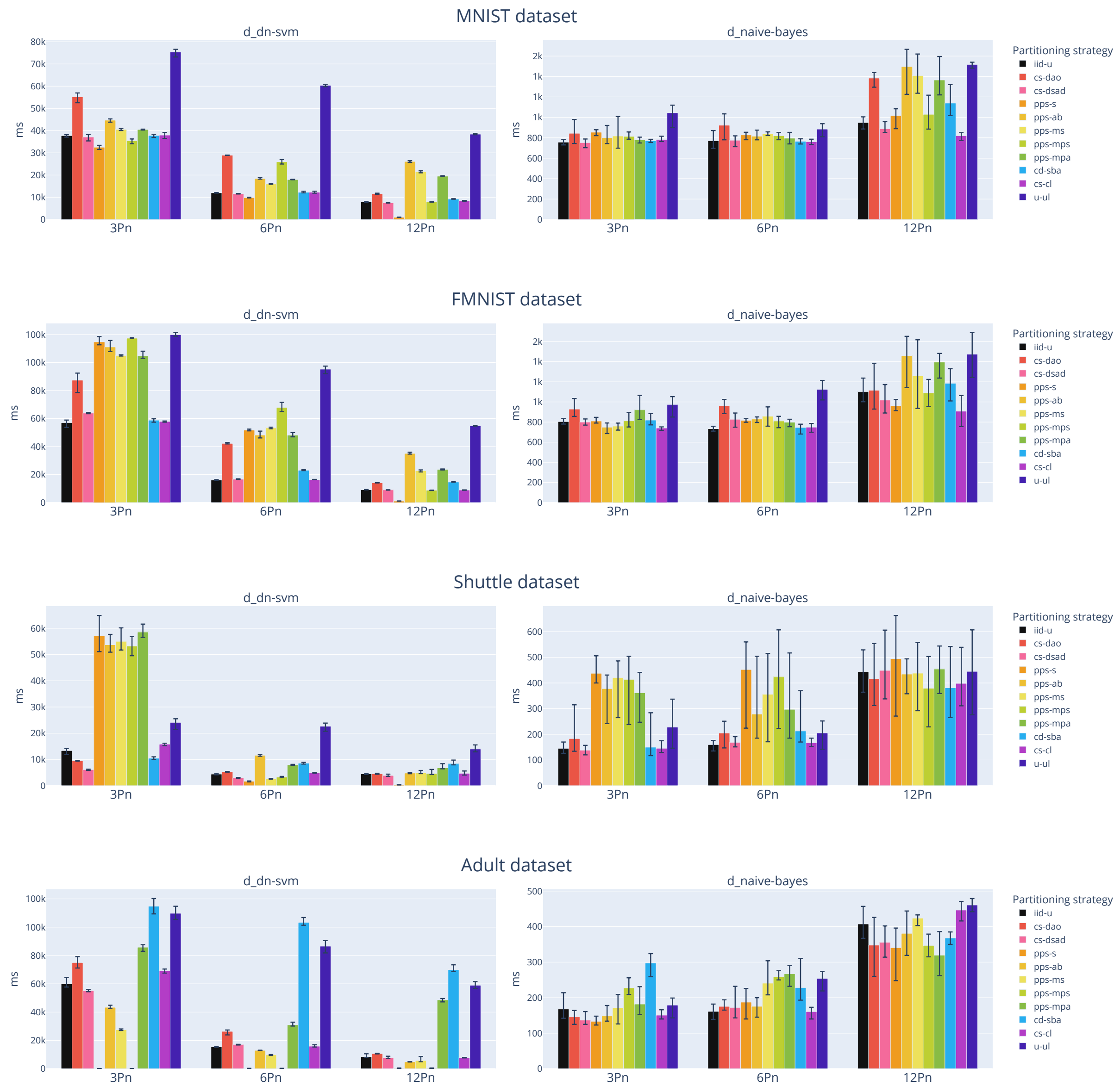


Figure 7.4: Detailed processing time statistics for classification evaluation.

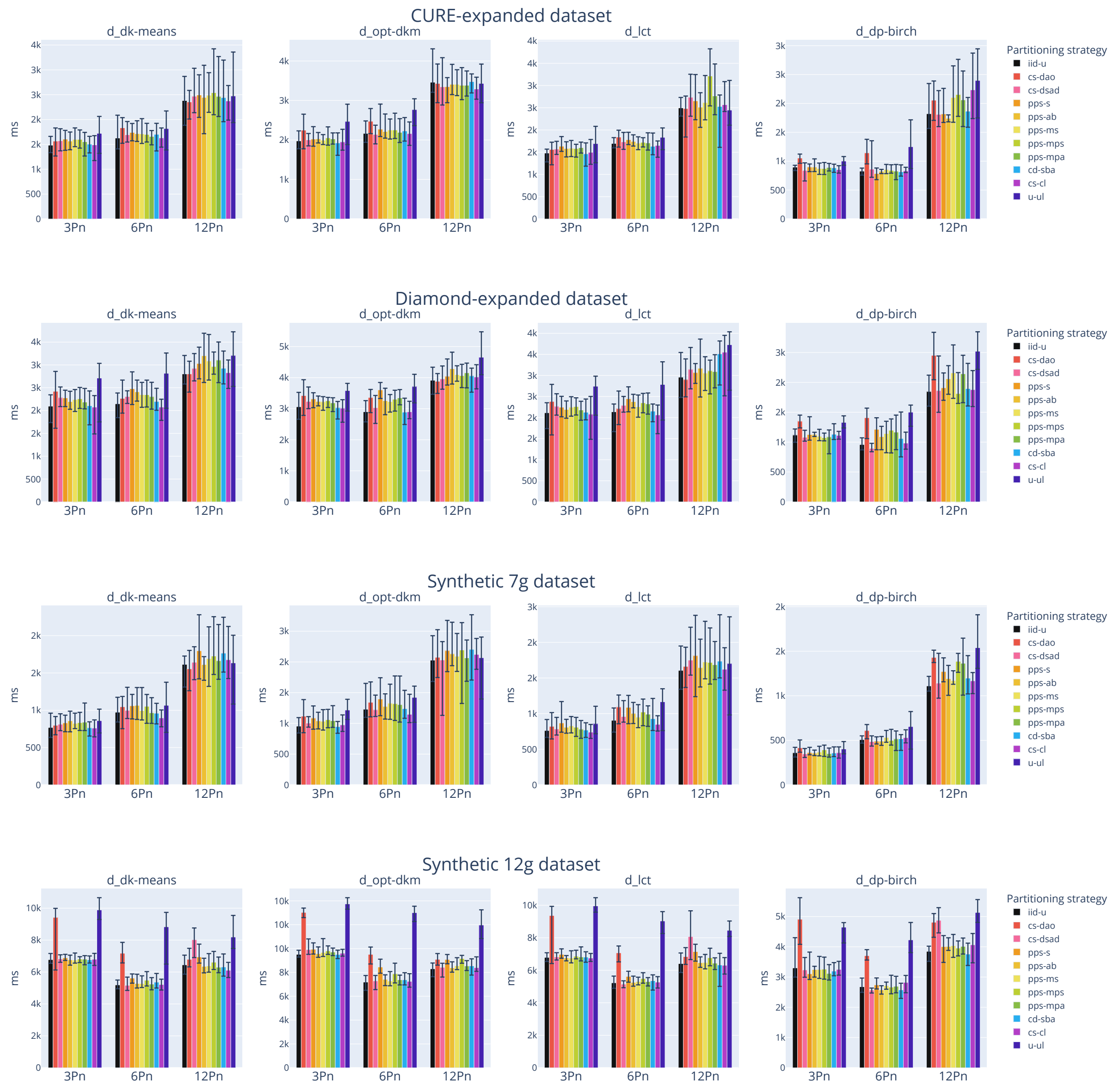


Figure 7.5: Detailed processing time statistics for clustering evaluation.

7.4 Detailed transfer load statistics

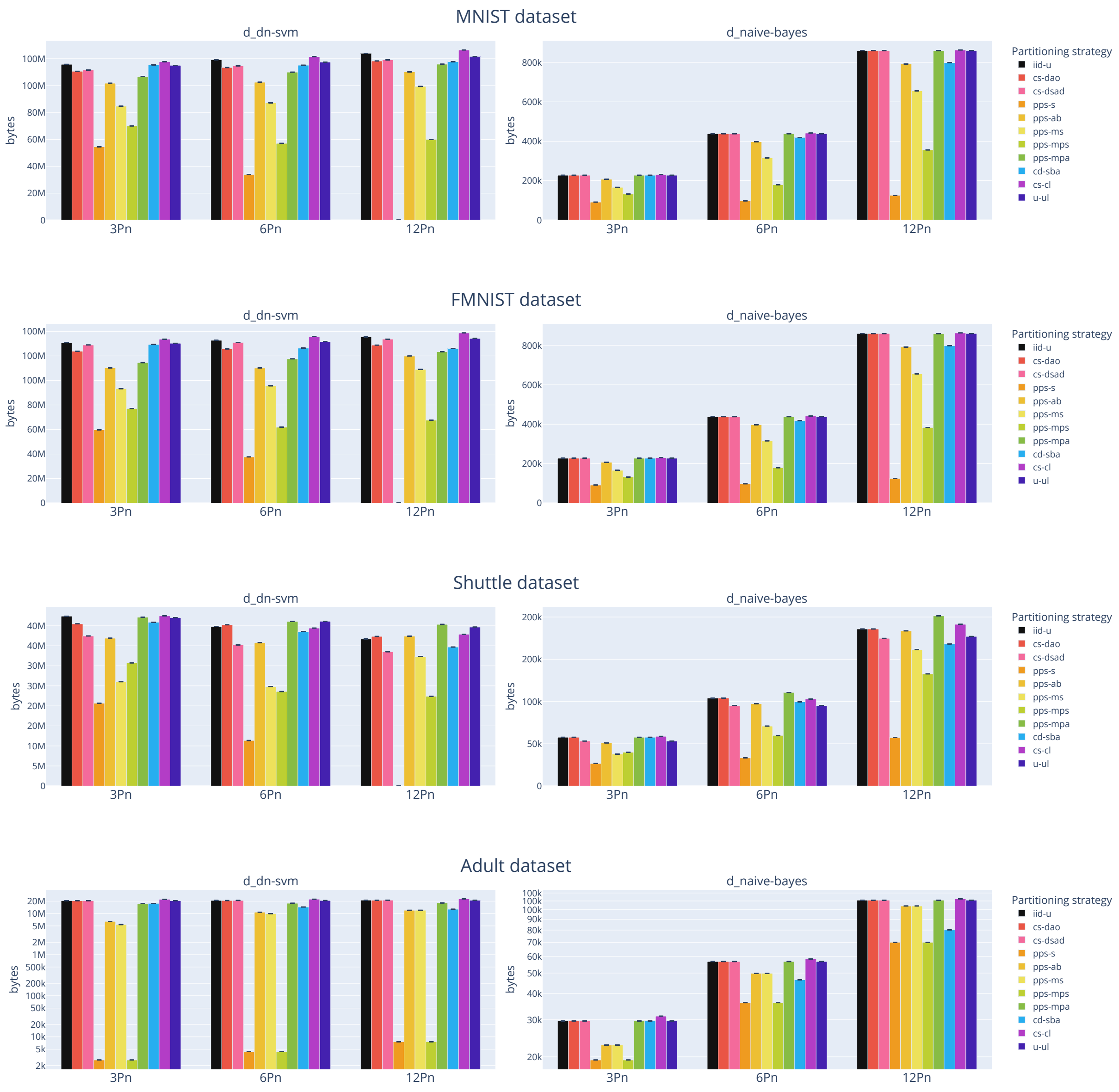


Figure 7.6: Detailed transfer load statistics for classification evaluation.

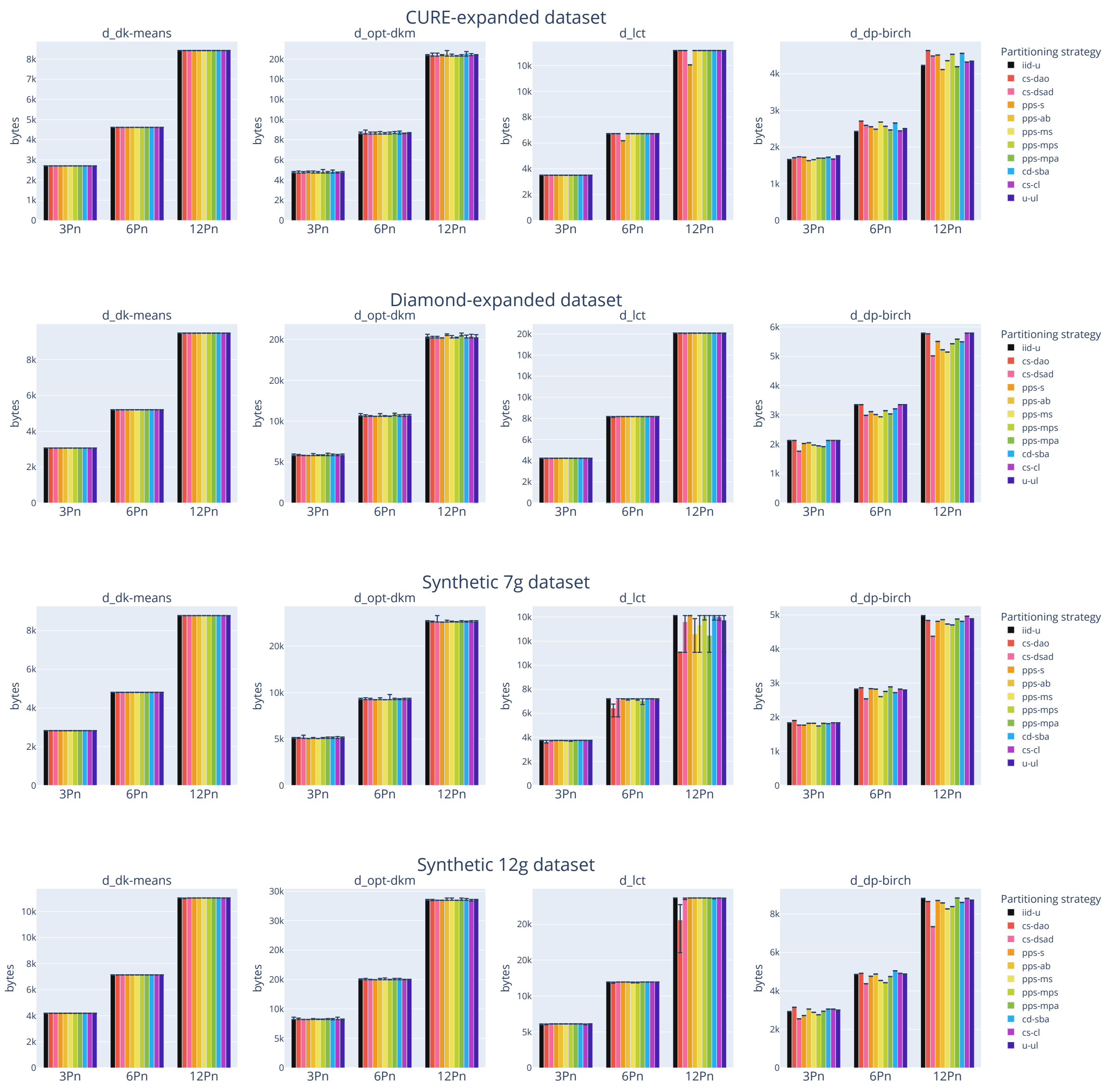


Figure 7.7: Detailed transfer load statistics for clustering evaluation.