Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Master Thesis

Data-driven Partitioning of Training Data for Complex Multiclass Problems

Tushar Rajendra Balihalli

Course of Study:	Computer Science
Examiner:	PD Dr. rer. nat. habil. Holger Schwarz
Supervisor:	Dennis Tschechlov, M.Sc.
Commenced:	January 5, 2022
Completed:	July 5, 2022

Abstract

Substantial technological advancements in the modern era have paved the way for the growth of a humongous amount of data that can be used for data analysis and decision-making processes. Data analysis tasks typically employ machine learning algorithms on real-world data. However, data in real-world scenarios contain a variety of complex characteristics like missing features, multi-class imbalance, and so on. Therefore, directly applying the machine learning methods does not lead to satisfactory results. As a result, data has to be pre-processed, e.g., by partitioning the data to reduce the complexity, before performing actual data analysis activities.

The objective of this work is to develop a partitioning approach using clustering that reduces the complexity of the challenges. To this end, various measures that can reflect the impact of the challenges are analyzed in detail. These measures quantify the complexity associated with the data. The focus of characteristics related to data complexity is focused on two major challenges: C1 and C2. Challenge C1 focuses on multi-class imbalance characteristic including high number of classes, overlapping decision boundaries, whereas challenge C2 comprises of heterogeneous feature characteristics involving missing features, sub-concepts and class membership problem. Although there are measures to address individual problems, this work focuses on addressing all of the challenges in a single dataset, thereby overcoming the shortcomings of approaches that address only one characteristic. The training data is subjected to data-driven partitioning using clustering, which is then optimized for the value of complexity measure. AutoML, an automated machine learning concept is employed for the hyper-parameter optimization. Further, a classifier is trained on individual partitions, and its hyper-parameters are optimized to improve the model's performance. The comprehensive evaluation discusses the results for different complexity measures and various state-of-the-art approaches using numerous validation datasets. The evaluation unfolds that partitioning of data with complex characteristics and optimizing for appropriate value of complexity measure increases system performance. Hence, this work demonstrates that the application of classification models on individual partitions aid for better performance in terms of prediction accuracy.

Kurzfassung

Erhebliche technologische Fortschritte in der heutigen Zeit haben den Weg für das Wachstum riesiger Datenmengen geebnet, die für Datenanalysen und Entscheidungsprozesse genutzt werden können. Bei der Datenanalyse werden in der Regel Algorithmen des maschinellen Lernens auf reale Daten angewendet. Die Daten in realen Szenarien enthalten jedoch eine Vielzahl komplexer Eigenschaften wie fehlende Feature-Werte, Ungleichgewicht zwischen mehreren Klassen usw. Daher führt die direkte Anwendung von Methoden des maschinellen Lernens meist nicht zu zufriedenstellenden Ergebnissen. Aus diesem Grund müssen die Daten vor der eigentlichen Datenanalyse vorverarbeitet werden, z. B. durch Partitionierung der Daten, um die Komplexität zu reduzieren.

Ziel dieser Arbeit ist es, einen Partitionierungsansatz mit Hilfe von Clustering zu entwickeln, der die Komplexität der Herausforderungen reduziert. Zu diesem Zweck werden verschiedene Maße, die die Auswirkungen der Herausforderungen widerspiegeln können, im Detail analysiert. Diese Maße quantifizieren die mit den Daten verbundene Komplexität. Der Schwerpunkt der Eigenschaften, die mit der Datenkomplexität zusammenhängen, liegt in dieser Arbeit auf zwei Herausforderungen: C1 und C2. Herausforderung C1 konzentriert sich auf die Eigenschaft des Ungleichgewichts zwischen mehreren Klassen, insbesondere bei einer hohen Anzahl von Klassen und überlappenden Entscheidungsgrenzen, während Herausforderung C2 heterogene Merkmalseigenschaften umfasst, welche u.a. fehlende Feature-Werte, Unterkonzepte und das Problem der Klassenzugehörigkeit beinhalten. Obwohl es Maßnahmen zur Bewältigung einzelner Probleme gibt, konzentriert sich diese Arbeit auf die Bewältigung aller Herausforderungen in einem einzigen Datensatz und überwindet damit die Unzulänglichkeiten von Ansätzen, die nur ein Merkmal berücksichtigen. Die Trainingsdaten werden einer Partitionierung durch Clustering unterzogen, welche dann für den Wert des Komplexitätsmaßes optimiert wird. AutoML, ein Konzept für automatisiertes maschinelles Lernen, wird für die Optimierung der Hyperparameter eingesetzt. Anschließend wird ein Klassifikator auf einzelnen Partitionen trainiert, und seine Hyperparameter werden optimiert, um die Leistung des Modells zu verbessern. In der umfassenden Evaluation werden die Ergebnisse für verschiedene Komplexitätsmaße und verschiedene State-of-the-Art-Ansätze anhand zahlreicher Validierungsdatensätze diskutiert. Die Evaluierung zeigt, dass die Partitionierung von Daten mit komplexen Eigenschaften und die Optimierung für einen geeigneten Wert des Komplexitätsmaßes die Systemleistung erhöht. Diese Arbeit zeigt also, dass die Anwendung von Klassifizierungsmodellen auf einzelne Partitionen zu einer besseren Leistung in Bezug auf die Vorhersagegenauigkeit führt.

Contents

List of Figures

List of Tables

1 Introduction

With rapid increase in the amount of data generated worldwide, there exists tremendous scope in the field of data analytics. It is estimated that data generated globally increases by approximately 23% per year [1]. This data can be efficiently utilized in the manner to extract relevant information that can be used in the processes of industry and research [KBT11]. Data mining techniques typically involve application of machine learning algorithms such as classification on top of data to carry out data analysis tasks. However, complexity of the data and its characteristics lead to challenges for data analysis while using classification methods.

The performance of classification methods depends on factors such as algorithm used, hyper-parameter configurations, characteristics of the dataset. For example, it is uncomplicated to perform data analysis activities on a dataset that contains a lot of data samples for each of the classes. However, this is not the case when it comes to real-world scenario, where data might not be present in the manner the data analyst wishes to be. The data might consist of characteristics such as presence of multiplicity of class labels that occur in an imbalanced manner leading to multi-class imbalance problem [HG09] [Wan11]. In such cases, machine learning algorithms tend to ignore class labels that occur less frequently. Furthermore, product variants with different physical properties might even lead to increase in the heterogeneous behaviour of the data in feature space [HRM20] [HRKM18]. For example, individual product variants have different value ranges representing different class patterns. This makes it difficult to identify distinguishable patterns in the data. Such characteristics would lead to challenges for data analysis. Direct application of data mining methods on such data would result in poor results thereby degrading the performance.

Literature typically addresses one of these above-mentioned characteristics in isolation. For example, an imbalanced class distribution can be addressed by techniques like over-sampling, under-sampling, which includes removing or adding random samples to make the dataset more even in terms of data proportion [MRA20]. Yet, these techniques do not address the characteristics related to heterogeneous behaviour. Although, the literature focuses on dealing with these individual data characteristics, the crucial challenge arises when multiple such characteristics occur together in the data [DZLL21]. The application of aforementioned techniques would have drawbacks when dealing with one of the characteristics, where they pose serious problems making data more complex in terms of other data properties. For instance, over- or under-sampling reduces the effect of class imbalance, but increases the problem posed by the large number of complex patterns per class [Gan12].

Recent research has shown that partitioning the data can help reduce the complexity posed by data characteristics [HRM20]. The work takes domain knowledge into account that can help solve the complexity challenge. By appropriately partitioning the data as per the domain knowledge, classification algorithms can then be applied on each of the partitions individually. Considering

[1] https://www.statista.com/statistics/871513/worldwide-data-created/

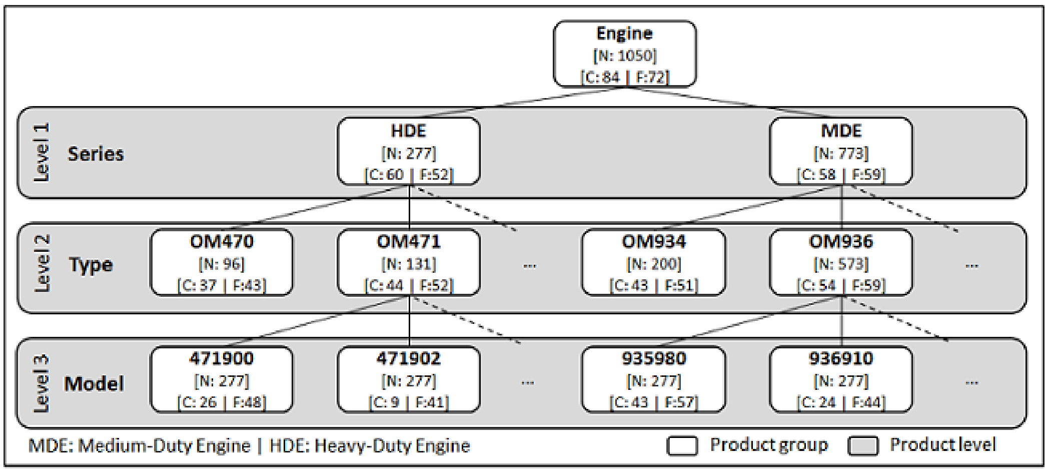


Figure 1.1: Partition of engine dataset [HRM20]

Figure 1.1, the engine dataset can be partitioned into various levels based on the domain knowledge. Classification algorithms for data analysis can then be applied on those individual partitions formed. This enables higher prediction accuracy when compared to the traditional approach of applying the classification on overall dataset. However, the approach requires domain knowledge that is not always available for the real-world data. As a result, there arises a need to develop a data-driven technique for partitioning training data so that the complexity of each partition decreases, allowing analysis methods to be applied to it. Clustering can be exploited to perform data-driven partitioning with the goal to reduce complexity of a subsequent classification task on each isolated partition.

While there are numerous data analysis techniques being proposed, the search for the optimal combination of algorithms and hyper-parameters remains an unsettling problem. This poses a particular challenge as there is no much literature available for AutoML for classification models [WTMH21] [KHS+21]. Therefore, AutoML4Clust [TFS] could be employed to achieve effective clustering results based on a predefined optimization metric. As there is a need to reduce the complexity for classification methods, it becomes evident to investigate complexity measures as optimization criterion. Data complexity measures [HB02] evaluate the dataset's complexity by taking into account the variety of patterns in each class. Such metrics could be optimized to address multiple challenges in combination.

The goal of this work is to examine how data-driven approaches for partitioning the training data can be exploited to address the predefined challenges C1 and C2. The contributions of this work include the following:

- Examination of different metrics to address the challenges C1 and C2 for data-driven clustering
- Investigation of clustering-based classification techniques in combination with complexity measures
- Prototypical implementation of examined metrics into classification pipeline based on the clustering results
- Evaluation of prototypical implementation, comparison of results with existing state-of-art approaches, hierarchy-based approach, random forest approach and complexity metrics-based approach

The rest of this work is organized as follows:

Chapter 2 provides the necessary background for this work. The chapter discusses various methods for clustering, classification using random forest, hyper-parameter optimization technique, and performance metrics. Chapter 3 discusses on related work along with existing solutions. Chapter 4 focuses on the proposed solution to the problem, describing how data-driven approaches can be used to reduce data complexity. It also discusses the use of various metrics and AutoML concept in an attempt to reduce complexity. Chapter 5 describes the concepts' implementation in detail. A comprehensive evaluation of data-driven partitioning for complex multi-class imbalanced datasets is performed in Chapter 6. This work is concluded in Chapter 7, which also includes an outlook for future work.

2 Background

This chapter covers the necessary background for this work. The clustering algorithms and metrics used in this thesis are presented in Section 2.1. Section 2.2 explains the classification model employed in the proposed approach. Section 2.3 defines the hyper-parameter optimization technique. The performance metrics are described in Section 2.4.

2.1 Clustering

Clustering [Jai10] is an unsupervised machine learning technique that groups data into a number of groups. Unlike classification algorithms, which have truth labels, clustering techniques categorize data samples without the presence of class labels, solely based on the data characteristics. This approach is used widely in many applications related to image segmentation, voice mining, research related to biology, etc. [Bij13]. The method aims to segment data samples such that similar samples are present in same group and dissimilar samples are present in different groups. In order to do so, a distance metric such as Euclidean distance [JD88] is employed to find the similarity between data samples. Various clustering algorithms exist that divide the data into partitions, of which the relevant ones for this thesis are described in the following. The section also presents the clustering metrics being used in this thesis.

2.1.1 Clustering Algorithms for Data Partitioning

Clustering algorithms are generally used in data partitioning because of their effectiveness [Jai10]. Typically, clustering algorithms split data into ‘k’ clusters based on certain criteria. However, the value of ‘k’ needs to be chosen by the user. Choosing the right value of ‘k’ is a complex task and various methods exist that estimate the ‘k’ value [DB79] [SJ03] [MMM13]. Four such popular clustering algorithms used in data mining applications are described below [VSC+12].

K-Means

The K-Means [HW79] algorithm clusters data into a pre-defined number of partitions based on the distance metric. The center of each partition (cluster) is referred as a centroid. The working procedure of K-Means is as follows:

1. Define the value of ‘k’ and number of iterations the algorithm needs to run.
2. Choose ‘k’ data samples as cluster centers in random.
3. Assign remaining data points to the nearest cluster center considering distance metric.
4. Re-compute the cluster centroid as the geometric mean of all cluster points.

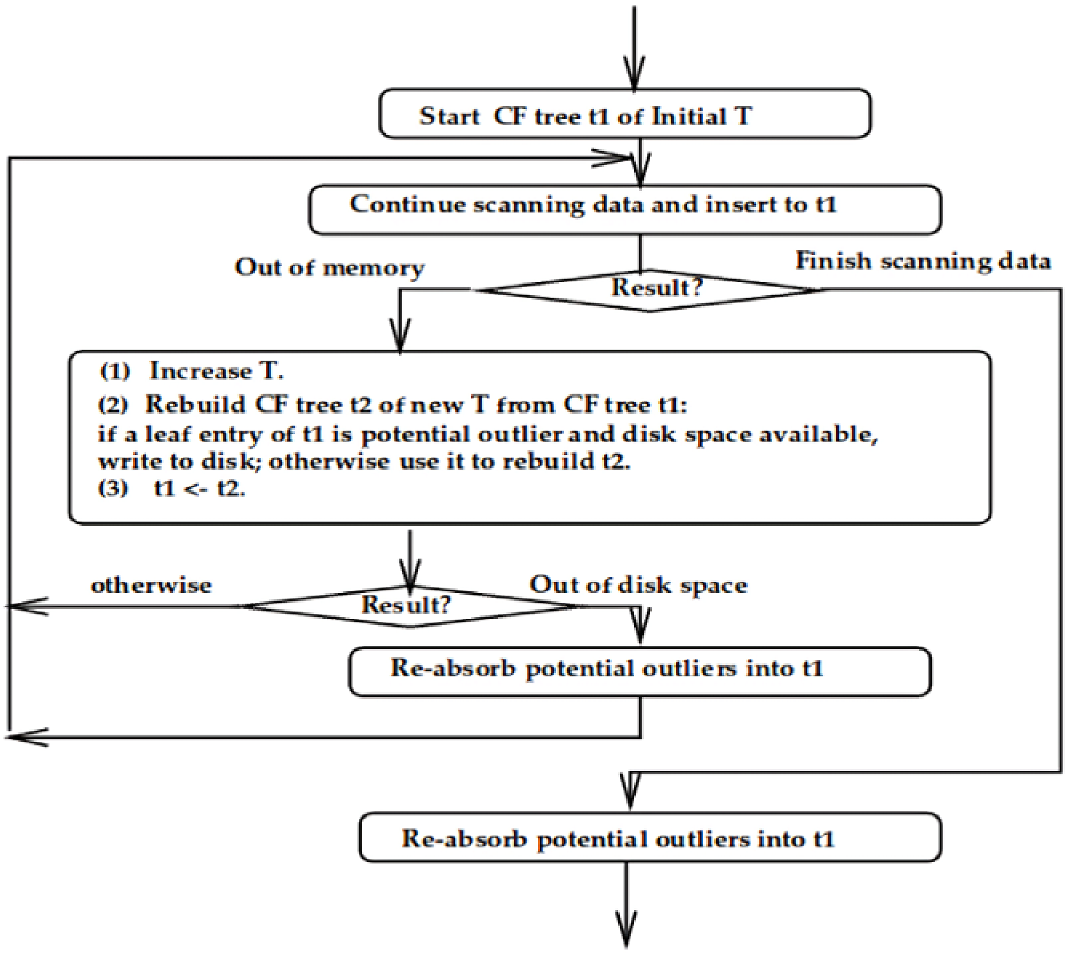

Start CF tree t1 of Initial T
Continue scanning data and insert to t1
Out of memory
Result?
Finish scanning data
(1) Increase T.
(2) Rebuild CF tree t2 of new T from CF tree t1:
if a leaf entry of t1 is potential outlier and disk space available,
write to disk; otherwise use it to rebuild t2.
(3) t1 <- t2.
otherwise
Result?
Out of disk space
Re-absorb potential outliers into t1
Re-absorb potential outliers into t1

Figure 2.1: Working principle of BIRCH algorithm [ZRL96]

5. Repeat steps 3 and 4 until there is no change in the cluster centroid, or the iterations threshold is reached.

Major disadvantage of K-Means is that it is sensitive to outliers. Also, the performance depends on the samples being chosen initially as cluster centroids. Because the initial centroids influence the final result, it is possible that it will end up in a local optimum. However, advanced methods exist that overcome these problems [AV06].

BIRCH

BIRCH [ZRL96] algorithm aims to partition data on large datasets using the hierarchical approach. BIRCH stands for Balanced Iterative Reducing and Clustering Using Hierarchies. It does not consider each data sample, rather provides the summary of all samples in a single scan. As a result, in the case of very large datasets, it can also be used in batches to update summaries in each iteration. A cluster feature (CF) tree is created that contain summary of clusters formed in the form of triplet (N, LS, SS), where N is the number of samples, LS is the linear sum of feature values, and SS is the squared sum of feature values. Brief working principle of BIRCH clustering is shown in the Figure

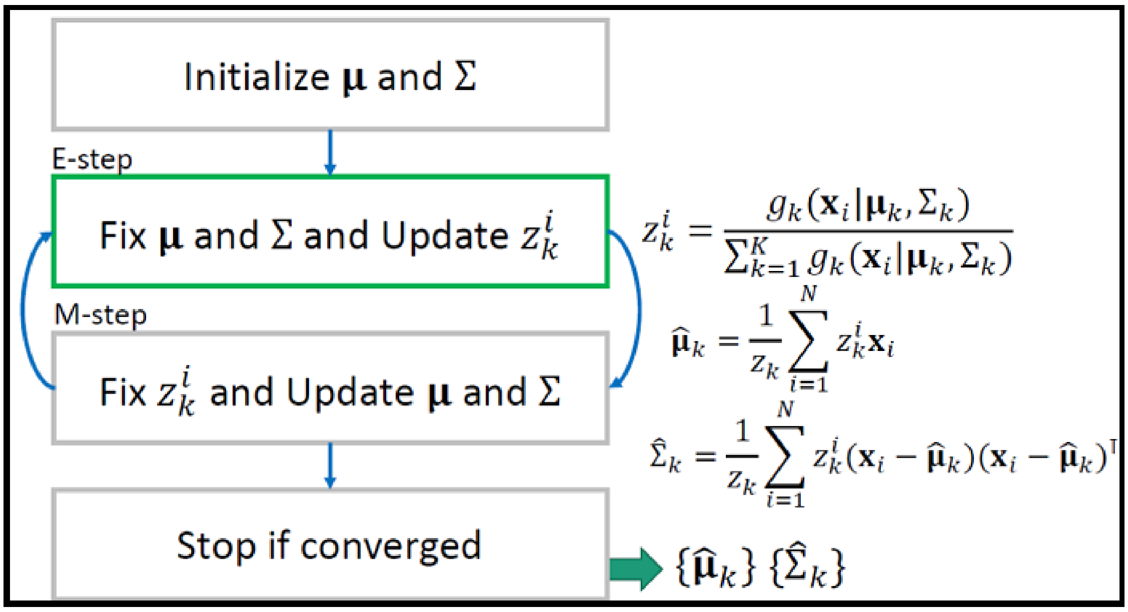

Initialize μ and Σ
E-step
Fix μ and Σ and Update z_k^i
M-step
Fix z_k^i and Update μ and Σ
Stop if converged
$z_k^i = \frac{g_k(\mathbf{x}_i|\mu_k,\Sigma_k)}{\sum_{k=1}^K g_k(\mathbf{x}_i|\mu_k,\Sigma_k)}$
$\hat{\mu}_k = \frac{1}{z_k}\sum_{i=1}^N z_k^i \mathbf{x}_i$
$\hat{\Sigma}_k = \frac{1}{z_k}\sum_{i=1}^N z_k^i(\mathbf{x}_i - \hat{\mu}_k)(\mathbf{x}_i - \hat{\mu}_k)^T$
$\{\hat{\mu}_k\}\ \{\hat{\Sigma}_k\}$

Figure 2.2: Working principle of GMM algorithm

2.1. It starts with creation of a CF tree structure where the summaries of data samples are updated as CF triplets. Once the memory of a CF tree is exhausted, the technique creates another CF tree to further absorb more data samples.

Gaussian Mixture Models

The Gaussian Mixture Models (GMM) [Ras03] uses the concept of Expectation-Maximization (EM) [Moo96], that assumes each partition to follow a Gaussian distribution N(x|μ,Σ). Here, μ represents the mean and Σ the standard deviation of the distribution of samples within each partition. Finally, a probability distribution π_i is calculated that describes the probability of a data sample x_i belonging to a specific partition k using the formula:

$$\pi_i = z_k^i = \frac{g_k(x_i|\mu_k,\Sigma_k)}{\sum_{k=1}^K g_k(x_i|\mu_k,\Sigma_k)}$$

where $g_k(x_i|\mu_k, \Sigma_k)$ is the probability function that considers mean μ and standard deviation Σ while assigning x_i to the partition k.

The Figure 2.2 illustrates the working of GMM. Each of ‘k’ clusters are initialized by a gaussian mixture consisting of (π, μ,Σ). The expectation step evaluates the probability π of a sample considering the current values of μ and Σ. The maximization step computes maximum likelihood estimation (MLE) thereby updating parameters μ and Σ. The E-M steps execute until convergence or a stopping criterion is fulfilled.

Bisecting K-Means

The bisecting K-Means [SB01] is a divisive hierarchical clustering algorithm that uses K-Means to cluster data samples. It overcomes the limitation of regular K-Means of recognising clusters of any size and shape. It initially assigns whole data to one cluster, and further splits the data based on an error criterion. Similar to regular K-Means, the parameter ‘k’ needs to be chosen by the user.

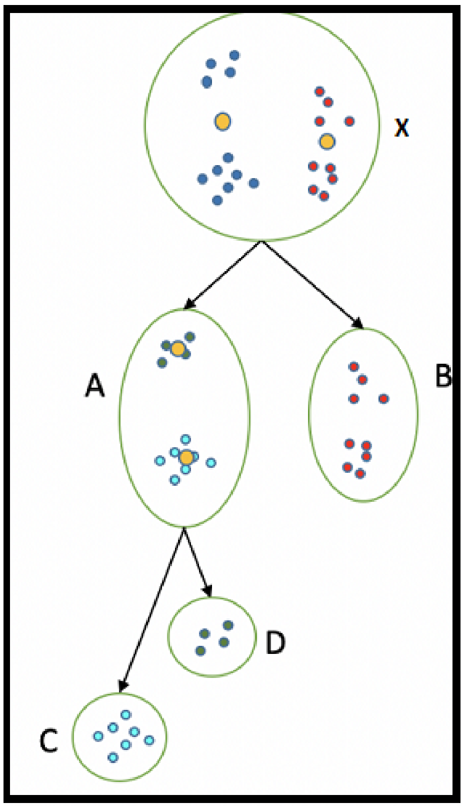


Figure 2.3: Example: Bisecting K-Means algorithm

The Figure 2.3 illustrates an example of bisecting K-Means algorithm. It starts with having all data samples within one big cluster, that is depicted by the whole data X. Subsequently, the algorithm uses K-Means with k=2 to divide the cluster into two clusters (A and B). Further, the cluster having the largest distortion is considered and bisected again using K-Means with k=2 (D and E). This procedure is repeated until the chosen 'k' clusters are formed. Distortion factor is computed using the squared sum of errors (SSE).

2.1.2 Clustering Metrics

Clustering results can be evaluated using a suitable metric. There are two different classes of metrics that evaluate clustering performance, namely internal and external metrics [JD88] [HBV01]. Internal metrics assess the internal structure of the clusters. They compute the compactness and the separation of the resulting clusters. External metrics require external information to evaluate the clustering results. Therefore, they compare the resulting cluster information with the expected clustering result. Because there are no expected ground truth labels in the proposed work, the clustering results are evaluated using internal metrics, which are explained below.

Davies-Bouldin Index (DBI): DBI metric compares the similarity of samples within clusters [DB79]. It is calculated by a quotient of a compactness and a separation measure. The compactness is calculated by the average distance of each point in a cluster to its centroid. The separation measure is computed by the distance between the centroids of the clusters. Lower values indicate better clustering results.

Calinski-Harabasz Index (CH): CH metric evaluates the variance ratio criterion [CH74]. It is measured by the ratio of sum of between-clusters dispersion to the sum of inter-cluster dispersion. Higher values of the CH index indicate denser clusters and thus better clustering.

Silhouette Coefficient (SIL): SIL metric measures the similarity of a data sample to its own cluster compared to other clusters [Rou87]. It is measured using the mean intra-cluster distance and the mean nearest cluster distance for each sample. Higher score indicates higher similarity of the data sample belonging to the cluster and hence better clustering.

2.2 Random Forest

While unsupervised machine learning methods like clustering work with unlabeled data, classification algorithms work with labeled data, making them supervised learning methods. In the current thesis, clustering algorithms are used to partition the training data. Following the partitions, a classifier is modeled on each partition. Ensemble classification is a popular data mining technique that uses a set of classifiers to predict previously unseen data samples. Such classification techniques perform well with multi-class problems. As a result, random forest, an ensemble learning method can be used for the classification tasks [Bre96]. It works on the concept of bagging and bootstrapping [Gra04], where multiple decision trees are generated using samples drawn at random from the data. Each of the individual decision trees in the ensemble provide a class prediction and the class with most votes is defined as the model's prediction. While constructing a decision tree, random subset of features and data samples are considered to ensure low correlation between all trees. Thus, a large number of relatively uncorrelated decision trees operating in an ensemble outperform any individual constituent models [BS16].

Figure 2.4 shows an example that illustrates the basic working of random forest. Multiple decision trees are generated using the samples drawn at random (bootstrapping). Furthermore, in order to classify an unknown data sample, the sample is fed into the ensemble, where each decision tree returns a result. Finally, the results are aggregated and the class with the most votes is declared to be the class of the unseen sample.

2.3 Hyper-parameter Optimization

Machine learning models are typically composed of two types of parameters, namely, model parameters and hyper-parameters. Model parameters are estimated during the model training. For example, weights in neural networks, coefficients in linear regression and so on. Hyper-parameters, on the other hand, are set prior to training the model. For example, the value of 'k' in K-Means, number of estimators in random forest and so on constitute the algorithms' hyper-parameters. Hyper-parameters define how the model is structured, hence the right set of hyper-parameters

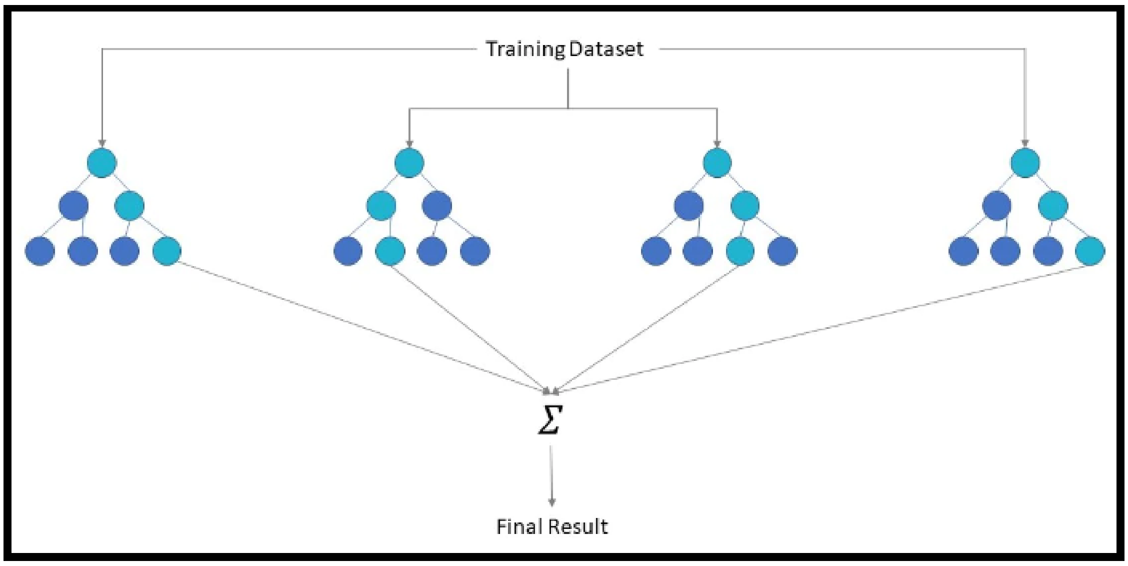


Figure 2.4: Example: Working of random forest

influences model's performance [FH19]. AutoML systems automate the task of applying machine learning to real-world problems [KHS+21]. Hyper-parameter optimization is regarded as the basic task of AutoML that reduces human effort necessary for applying machine learning. Several works exist that tackle supervised learning for AutoML [WTMH21], but less attention is given to clustering. Yet, in a recent work, AutoML4Clust [TFS] is developed that applies the AutoML concept to clustering method.

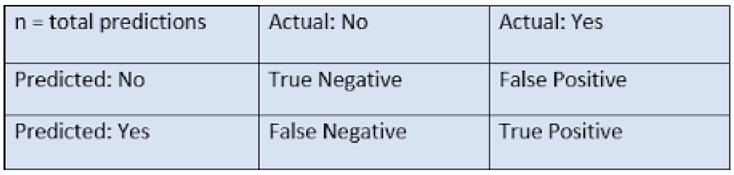
Various optimization techniques exist [WCZ+19] [BB12] [LJD+17] that aid in optimizing hyper-parameters in enhancing the performance of machine learning algorithms. The primary task of these techniques is to optimize a black-box function f, in which input is a configuration from the configuration space. A configuration space contains various hyper-parameters of a machine learning model along with their range. The common steps of the optimizers are as follows:

1. Select a configuration to execute.
2. Execute the current configuration on a data.
3. Evaluate the results of the configuration based on a metric.

Steps (2) and (3) are generally included in f. However, different techniques such as Bayesian optimization vary in how they perform step (1), i.e., in selecting a configuration. These steps are carried out for a fixed number of iterations, known as the budget.

2.4 Performance Metrics

Performance metrics assess how well a machine learning model performs in a given context. In terms of classification models, performance metrics quantify the quality of the model by comparing the predicted results from a classifier model with the ground-truth labels. There are various



n = total predictions	Actual: No	Actual: Yes
Predicted: No	True Negative	False Positive
Predicted: Yes	False Negative	True Positive

Figure 2.5: Confusion matrix

metrics available that assess the performance of a model [Bot18] [EK21]. However, selecting the appropriate metrics is dependent on the type of model being used. As a result, to evaluate a classifier's performance, the following two metrics are described, which are used in this thesis.

Accuracy: Accuracy is the most popular metric in evaluating the performance of multi-class classification problems [GBV20]. It defines the degree to which the predicted results are correct. It is determined by the ratio of correctly predicted outcomes to all outcomes. Therefore, a higher value of accuracy indicates better model performance.

F1-score: F1-score is a measure of a model's accuracy on a dataset [GBV20]. It evaluates the performance of classification models efficiently, especially when dealing with imbalanced multi-class distribution [WY12]. The F1-score evaluates the model's performance using a confusion matrix [Tha20]. A confusion matrix visualizes and summarizes a classification algorithm's performance. It includes predicted and actual values, as well as the total number of predictions. The Figure 2.5 depicts the features of the confusion matrix, which are explained further below.

- **True Positive (TP)**: The model has predicted true, and the actual value is also true.
- **True Negative (TN)**: The model has given prediction false, and the actual value is also false.
- **False Positive (FP)**: The model has predicted true, but the actual value is false.
- **False Negative (FN)**: The model has predicted false, but the actual value is true.

Two other terms are extracted from the confusion matrix before computing the F1-score, namely, precision and recall . Precision indicates how good the model is predicting a category of a class, whereas recall indicates how many times the model was able to detect a category of a class. These terms are given by the formulae:

$Precision = \frac{TP}{(TP + FP)}$; $Recall = \frac{TP}{(TP + FN)}$

Thus, F1 is given by the harmonic mean of precision and recall. The formula for F1-score is given by:

$F1 = \frac{(2 \times precision \times recall)}{(precision + recall)}$

Since the F1-score indicates the model's predictive performance, a higher score indicates better classifier performance.

3 Related Work

The section related work introduces various types of complex characteristics present in the dataset, followed by measures that exist to overcome those complexities. An overview of existing complex characteristics is given in Section 3.1. Section 3.2 discusses measures that address one such type of complexity, i.e., multi-class imbalance. Section 3.3 deals with direct application of random forest classification method on the whole dataset, whereas Section 3.4 discusses about the technique that exploits domain knowledge requiring hierarchies to thereby address the complexity challenges.

3.1 Complex Data Characteristics

A real-world data typically consists of various complex characteristics that makes it difficult to directly apply data analysis methods. Before applying the data analysis techniques, it becomes crucial to examine what kind of challenges are incorporated within the data. The complexity of data stems from several characteristics, such as a highly imbalanced class distribution (C1) and a heterogeneous feature space (C2).

3.1.1 Multi-class Imbalance (C1)

Multi-class imbalance is one such characteristic where the dataset contains multiplicity of labels occurring in an imbalanced manner [WY12]. Figure 3.1 depicts the nature of imbalanced class distributions present in the dataset, where very few classes exist that contain higher number of instances and large number of classes have only a few class instances. This type of behaviour results in two types of class formations, namely majority classes (classes with many samples) and minority classes (classes having less samples), thus resulting in the behaviour of an imbalanced class distribution. In such scenarios, class labels that occur less frequently are ignored by the machine learning algorithms. In addition, presence of overlapping decision boundaries of the different classes induces further complexity within the data [Ste13]. Furthermore, the existence of multiple sub-concepts for each of the multiple classes would exacerbate the dataset's complexity [Kra16]. Figure 3.2 illustrates the notion of sub-concepts [HRM20] containing two classes circle and star. It can be seen that the class circle appears in more than one distinct set of ranges. The same holds true for the star class as well. As a result, multiple sub-concepts are often included inside the same classes, making the dataset more complex. Combination of aforementioned sub-challenges can be formulated under the characteristic C1.

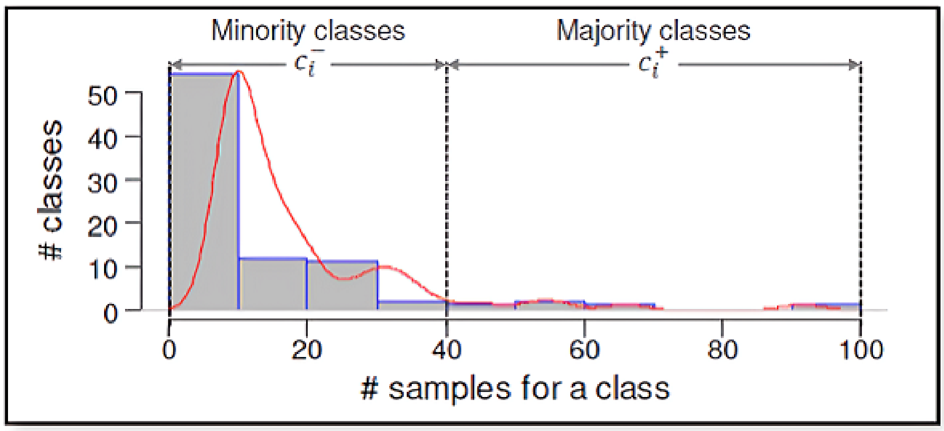


Figure 3.1: Example: Input class distribution of imbalanced data

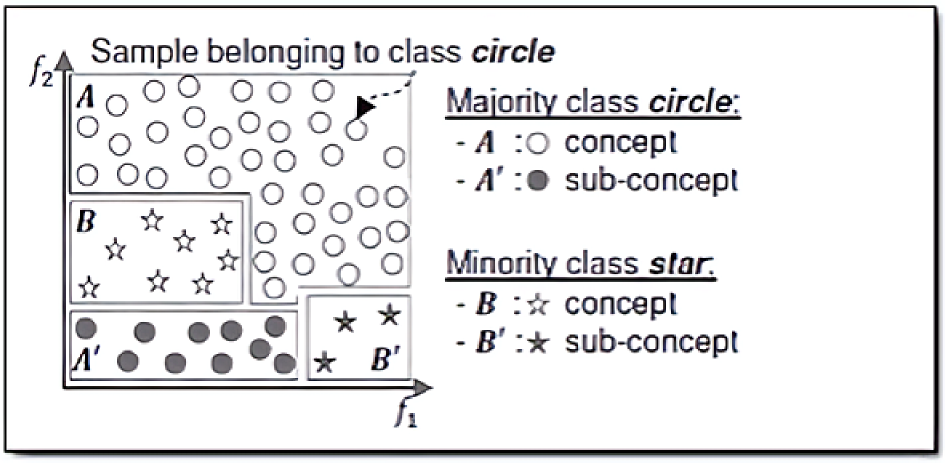


Figure 3.2: Example: Sub-concepts

3.1.2 Heterogeneous Feature Space (C2)

Further, real-world data is widely assumed to contain other casualties such as missing features and class membership issues, all of which can degrade data analysis performance [HRM20]. Presence of missing features would more likely cause errors when dealing with machine learning models. For example, considering the Figure 1.1, each cylinder of an engine transmits a sensor signal. Each of these signals is represented by a column in the overall data. In this scenario, a data sample for a four-cylinder engine version would include six cylinder columns, with two signals not holding a value. Hence these missing values of signals need to be either imputed or removed before the data analysis. Also, there might be possibilities that there exist different variants (levels) in the feature space, making data more complex. Thereby, it is even possible that not all features are present in those variants. For instance, considering the preceding scenario, sensor signals corresponding to cylinders No. 5 and No. 6 would be available in sample for a six-cylinder engine, but not in four-cylinder sample. As a result, not all classes are applicable to all data samples. Therefore, when

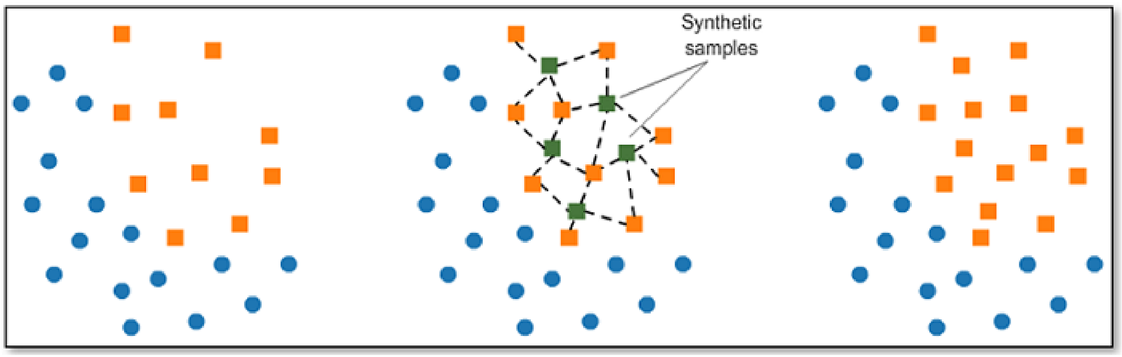


Figure 3.3: SMOTE: Example of the process

training a model, features that do not represent a specific set of classes must be isolated. Otherwise, it would result in inaccurate predictions. Combination of all these sub-challenges in data account for the characteristic C2.

3.2 Multi-class Imbalance Measures

This section focuses on measures being employed in order to address the multi-class imbalance problem associated with the data. While the majority of imbalance learning techniques deal with two-class problems [FLG+13] [SWK09] [HYS+17] [GFB+11], there are a few that are also effective for multi-class tasks [ZL10]. There also exists methods which convert a multi-class problem into several two-class problems and work with two-class imbalance techniques [WY12]. However, these techniques show reduced prediction performance in case of complex data characteristics [LD13]. Yet, some solutions are proposed below that focus on modeling with multi-class imbalance data in a straightforward manner.

3.2.1 Random Oversampling and Undersampling

One of the proposed techniques to deal with multi-class imbalance problem is by applying random oversampling or under-sampling [MRA20]. In order to cope up with unequal distribution of classes in the dataset, the technique deals with randomly adding samples to the minority classes or deleting certain number of data samples from the majority classes. According to the authors, oversampling performs slightly better than undersampling in terms of evaluation metrics.

3.2.2 Synthetic Minority Oversampling Technique (SMOTE)

The technique SMOTE was proposed as a substitute for random oversampling and undersampling procedures [CBHK02]. Figure 3.3 illustrates the idea of SMOTE. This involves randomly picking data points from minority classes and computing the k-nearest neighbours for the point. Synthetic points are then added between the chosen data sample and its neighbours. These synthetically added points increase the proportion of data samples from minority classes. Class orange squares

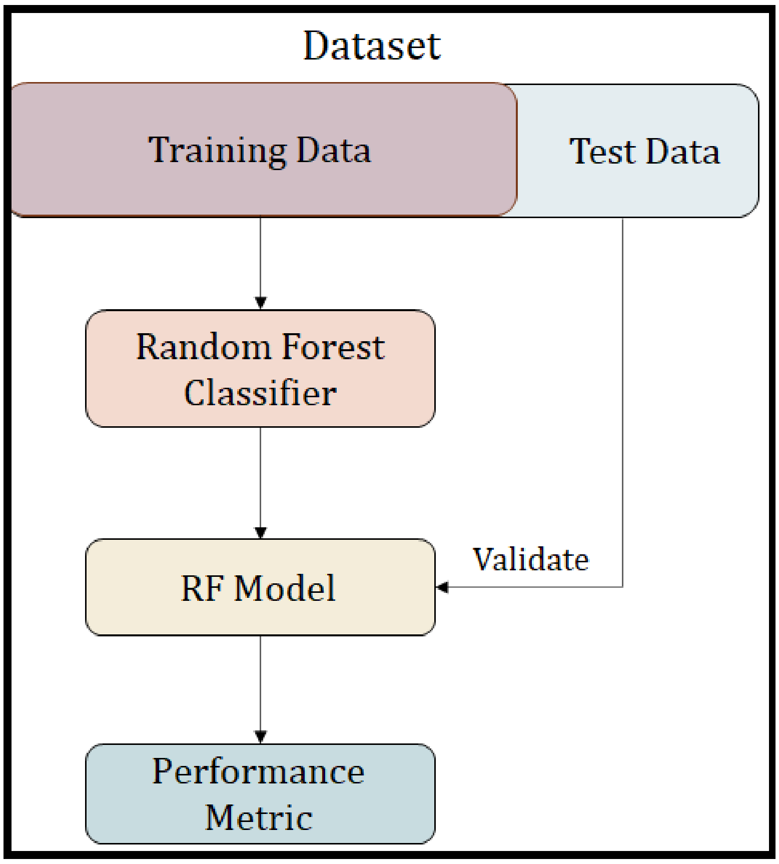


Figure 3.4: Flow Diagram: Random forest baseline approach

that constitute to minority class is being selected and random synthetic points are added (green data samples in the figure) considering the neighbours. These would then function as normal data points, addressing the issue of class imbalance.

3.3 Ensemble Learning Method

Ensemble learning methods can be used to address the problem of multi-class imbalance [VD20] [HYY+16]. These methods are considered to be more effective than data sampling techniques in improving classification performance of imbalanced data [FHR18]. An ensemble learning classifier combines several classifiers to determine the class label for newly unlabeled items. Random Forest [Bre96] is a popular ensemble learning method that can be utilized to address the problem of imbalanced data.

The procedure of employing random forest to address the predefined complex data characteristics is illustrated in Figure 3.4. Overall dataset is being split into training and testing dataset for modelling and validating purposes respectively. A random forest classifier (RFC) is then trained on the training dataset to create random forest ensemble model. This model would then be validated against test data in order to find performance metrics such as accuracy, confusion matrix, etc. The literature states that the RFC performs well on imbalanced dataset in terms of prediction accuracy [MR17].

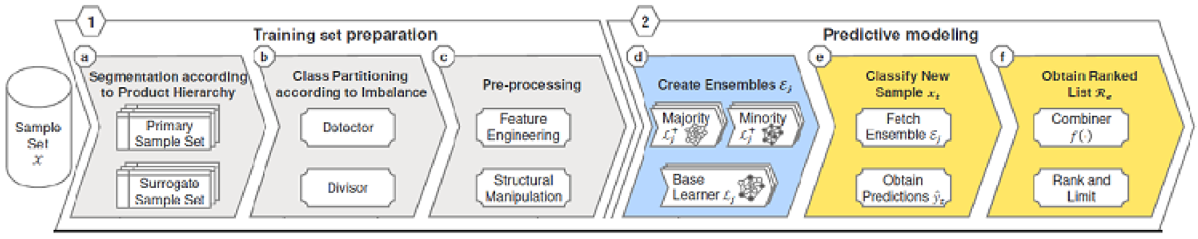

1
Training set preparation
a
Segmentation according to Product Hierarchy
Primary Sample Set
Surrogate Sample Set
b
Class Partitioning according to Imbalance
Detector
Divisor
c
Pre-processing
Feature Engineering
Structural Manipulation
Sample Set $\mathcal{X}$
2
Predictive modeling
d
Create Ensembles $\mathcal{E}_j$
Majority $\mathcal{L}_j^\dagger$
Minority $\mathcal{L}_j^\dagger$
Base Learner $\mathcal{L}_j$
e
Classify New Sample x_t
Fetch Ensemble $\mathcal{E}_j$
Obtain Predictions $\hat{y}_t$
f
Obtain Ranked List $\mathcal{R}_e$
Combiner $f()$
Rank and Limit

Figure 3.5: Flow diagram of classification system

Also, due to the fact that random forest creates ensemble of decision trees considering features and data instances in random fashion, it is likely to reduce the effect of over-fitting. Although, random forest can be applied on the whole dataset, the performance can still be improved by making use of clustering-based classification techniques [FTR14] [TTA17] [ZWGS10]. This method is being considered as one of the baseline approaches against which the proposed method will be compared.

3.4 Partitioning by incorporating Domain Knowledge

While literature focuses on solving individual analytical challenges in isolation, it neglects to address the combination of both challenges C1 and C2. Therefore, Hirsch et al. proposed an approach that exploits domain knowledge in order to partition the training data in a systematic manner [HRM20]. The proposed system incorporates domain knowledge from a product hierarchy to segment the training data set into several subsets (Figure 1.1). As a result, each such subset contains data samples that are technically similar in terms of product variants. The authors argue that the classification system has to consider an appropriate algorithm that takes care of different analytical challenges and their mutual influences together. Classification algorithms being modelled on each of the subsets yields better performance since the feature space and data instances within one subset are as similar as possible, thus maintaining homogeneity across individual partitions.

Figure 3.5 depicts the flow diagram of the proposed classification system. The procedure is divided into two stages: training set preparation and predictive classifier modelling. The first stage mainly involves application of two techniques, namely SPH and CPI. Segmentation of training data according to a known product hierarchy (SPH) is performed to remove the heterogeneity in data. It is followed by the technique of class partitioning according to imbalance (CPI) to address the imbalance issue. Furthermore, the stage related to classifier modelling focuses on fitting random forest ensembles on the subsets generated in the first stage. Predictions are then made on top of the validation dataset to produce a ranked list of predicted classes. Because the goal is to address challenges C1 and C2, the subsections that follow explain the techniques involved in detail, namely SPH and CPI. The final subsection summarizes the approach's remaining steps.

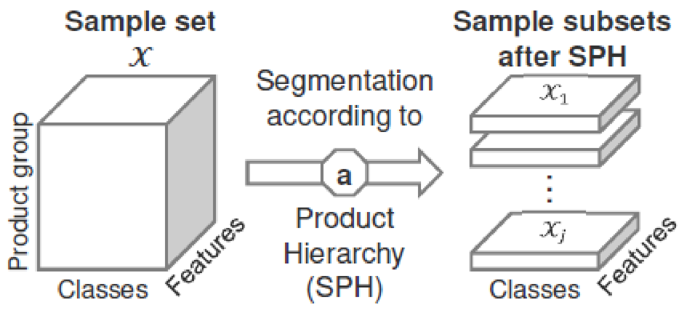


Figure 3.6: Segmentation strategy of SPH

3.4.1 Segmentation according to Product Hierarchy (SPH)

This stage aims to partition the data before applying classification algorithms for data analysis. Segmentation over training data is performed based on domain knowledge. A product hierarchy is created using knowledge of the domain to which the data belongs. Following that, the data is partitioned based on the hierarchy. Figure 3.6 illustrates the segmentation strategy of SPH method. As the training data is segmented systematically, the heterogeneity is reduced. However, it becomes evident to select a proper hierarchy level when partitioning the training data. As the level of hierarchy grows deeper, the number of data samples in each subset decreases. As a result, it would be unsuitable to train a classifier in case of fewer samples. Therefore, the author introduces the concepts of primary and surrogate sample subsets. Primary subsets are found at the bottom of the product hierarchy, while surrogate subsets can be found one level up. The system chooses primary subset as its partition if there are enough data samples to train a classifier, otherwise it chooses surrogate subset. This can be well understood using the example illustrated using Figure 1.1. The root dataset, 'Engine,' is broken down into its product hierarchy at several levels. However, at the lowest level, there are subsets with a much smaller number of data samples. In such circumstances, the data is analysed using surrogate subsets from one level above. Hirsch et al. argue that decomposing the dataset according to its product levels addresses the problem of sub-concepts [HRM20].

3.4.2 Class Partition according to Imbalance (CPI)

The goal of this stage is to separate the majority and minority classes that are present in the partitions after SPH. This stage aims to eliminate the class imbalance challenge once the system has decided whether to use primary or surrogate subsets based on the number of data samples present. As a result, class partitioning is performed on each partition to generate disjoint majority and minority subsets. This separation of majority and minority classes ensure that minority classes are not ignored when machine learning algorithms are applied to them.

However, the stage CPI is employed only when the class imbalance scenario exists. Hence, a check is required to detect the imbalance property in the subset. If all classes have sufficient samples, there is no need to use the CPI technique; instead, simply SPH is used. Figure 3.7 illustrates the strategy of applying CPI method on two subsets. CPI begins functioning on individual subsets after the dataset has been partitioned using the SPH technique. Considering subset 1, out of the four

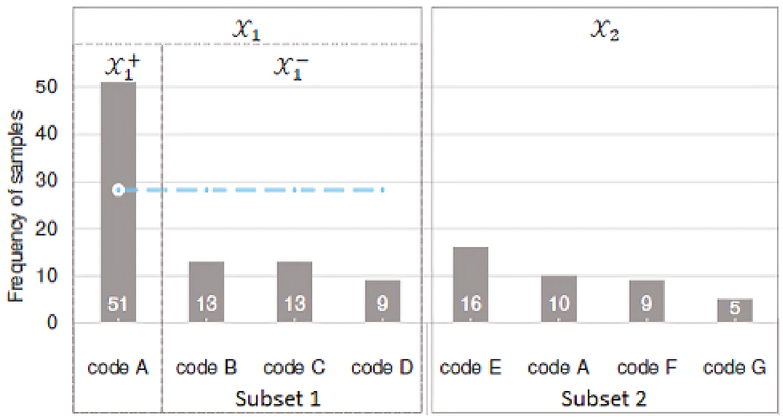


Figure 3.7: Example: CPI strategy

codes (classes) present, code A has very high number of samples compared to the codes B, C and D. Hence, a dynamic threshold is considered (blue margin line in the Figure 3.7) above which all classes are placed in the majority subset, while the remaining ones are placed in the minority subset. In this way, code A gets separated out from whole data X_1 into majority subset denoted by X_1^+ and the rest three codes B, C and D being put into minority subset denoted by X_1^-. This results in a comparable number of data samples within the same subsets. Machine learning algorithms such as classification methods when trained on these individual majority and minority subsets tend to perform better. Taking subset 2 into account, however, there is no situation in which the codes differ significantly in terms of number of samples. Therefore, there is no need to partition the subset using CPI.

3.4.3 Further steps

Once the SPH and CPI are performed on the training data, all the subsets are then checked for data quality in the pre-processing step. Since machine learning algorithms are sensitive to invalid values, the step aims to eliminate sparse, zero and near zero-variance features. Also, to enhance data quality, normalizing continuous values and performing one-hot encoding on categorical features are being also carried out.

Individual classifiers are then trained on individual subsets, whether they belong to majority X_j^+, minority X_j^- or just the whole subset X_j. Once the ensembles have been modelled, they are utilized to validate the predictions using test data. A ranked list is used to analyse performance indicators such as accuracy.

4 Data Partitioning Approach

This section presents a data-driven technique for partitioning training data in order to address the complex data characteristics. The data consists of **C** classes, **N** data samples and **F** features. The basic idea is to partition the data such that the complexity on each partition is reduced. Section 4.1 provides an overview of the approach. Section 4.2 introduces data complexity measures, which are used to quantify the classification complexity of data. Section 4.3 describes how optimization of data-driven clustering and complexity measures could be used to partition the training data for subsequent classification. Section 4.4 demonstrates the concept of using a random forest classifier on partitioned data. Finally, Section 4.5 aims at improving the performance of the proposed system by optimizing hyper-parameters of random forest classifier.

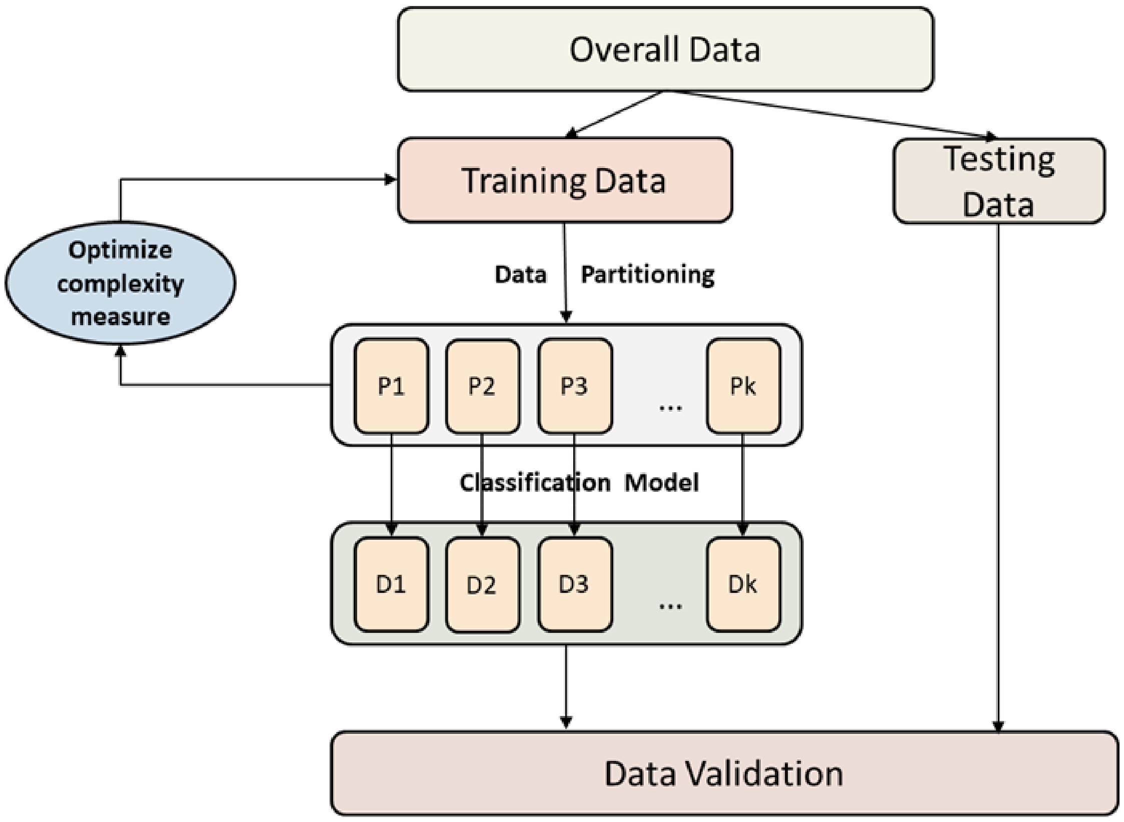


Figure 4.1: Overview diagram of proposed approach

4.1 Overview

While the majority of the methods proposed in the related work section aim to solve individual challenges in isolation, these approaches typically have drawbacks when compared to the other challenge. The approach based on domain knowledge [HRM20] aims to solve the challenges in a combined fashion. The main assumption of this approach, however, is the availability of domain knowledge that requires a product hierarchy. This assumption does not always hold in practice. In most cases, such a hierarchy is not pre-defined in each domain to which the data belongs. In such cases, there is a need to develop an approach that partitions the training data using a data-driven technique such as clustering. As a result, the proposed method intends to use data-driven clustering to segment the dataset in order to address the multi-class classification problem. Data complexity measures [HB02] assess the dataset's complexity by considering the variety of patterns present in each class. Such complexity measures can be optimized to solve the problem. Classification algorithms are modelled over individual partitions to improve system performance.

The overview diagram of the proposed approach is shown in Figure 4.1. The dataset is divided into two groups for training and testing. Further, the training data is partitioned using a data-driven clustering. Multiple clustering algorithms are executed and the complexity measures are optimized to obtain a low complexity on each partition. Classification algorithms are modeled to each of these partitions. These models are utilized to validate the system through the use of test data.

4.2 Data Complexity Measures

The idea behind the proposed method is to partition the training data such that the partitioned data has less complexity when compared to the overall data. Literature states that classification problems often face challenges due to the overlapping class distributions, sparsity or dimensionality of the data, and the complexity of the classification boundary [HB02]. These problems affect the performance of classifier models. To quantify this complexity in each partition, measures are required that compute how well classes are separated. Data complexity measures [HB02] highlight how classes are separated or interleaved, which is the most important factor that affects classification accuracy. These metrics are employed to measure the complexity associated with each partition. Thus, optimizing for the value of these complexity measures helps in reducing the overall complexity of the data. Furthermore, complexity measures can be applied to any type of classification data as they make no assumptions about the domain and thus can be used in any real-world situation.

Table 4.1 provides the list of data complexity measures used in this work. These are categorized according to the type of problem they address. The first category focuses on the issue of class overlapping, while the second category measures the complexity caused due to neighboring classes. Category related to dimensionality measures relate to the effect of curse of dimensionality on the complexity of data. Finally, the class balance category measures the imbalance degree of the data present in a partition. Each of the complexity measures is explained category wise in detail below along with their relation to the complexity.

1. **Feature Overlapping**: Feature overlapping category concentrates on the overlap of samples belonging to different classes. The overlap information is computed per feature. It consists of the following three measures:

Category	Description	Data Complexity Measure	Acronym
Feature Overlapping	Class overlap per feature	Fisher's Discriminant Ratio	F1
		Volume of Overlap Region	F2
		Feature Efficiency	F3
Neighborhood Measures	Nearest class data points	Fraction of Borderline Points	N1
		Ratio of intra/inter class nearest neighbor distance	N2
		Error rate of the nearest neighbor classifier	N3
Dimensionality Measures	Check for dimension reduction	Ratio of the PCA dimension to the raw dimension	T4
Class Balance Measures	Class distribution information	Imbalance Degree	ID

Table 4.1: Employed data complexity measures

Fisher's Discriminant Ratio (F1): This measure computes class overlap for a single feature. Considering one feature at a time, It computes the mean and standard deviation of different classes of data and investigates how distant each class samples are from each other. In general, F1 is given by the following formula:

$$F1 = \frac{\sum_{i=1}^{C} n_i.\delta(\mu, \mu_i)}{\sum_{i=1}^{C} \sum_{j=1}^{n_i} \delta(x_j^i, \mu_i)},$$

where n_i is the number of samples of class i, δ being the distant function given by Euclidean distance, μ being the overall mean, and μ_i denoting the mean of class i.

The F1 ratio represents how well a specific feature distinguishes samples from different classes considering overlapping. Thus, a higher F1 value indicates better class separation and, as a result, lower complexity.

Volume of Overlap Region (F2): The measure F2 computes the volume of overlapping region considering a single feature. Maximum and minimum values of each class are calculated, and the ranges of different classes are compared to see if there is any overlap. Further, the measure is normalized based on the feature's overall range. The formula to compute F2 is given by:

$$F2 = \prod_{i=1}^{d} \frac{\min(\max(f_i, c_1), \max(f_i, c_2) - \max(\min(f_i, c_1), \min(f_i, c_2)}{\max(\max(f_i, c_1), \max(f_i, c_2) - \min(\min(f_i, c_1), \min(f_i, c_2)},$$

where $max(f_i, c_j)$ and $min(f_i, c_j)$ are the maximum and minimum values of feature i in class j respectively, $i = 0, 1, .., d$ denotes the feature dimensions of the data.

Since F2 computes the overlap information of different classes, the value needs to be lower as possible, which accounts to lower complexity.

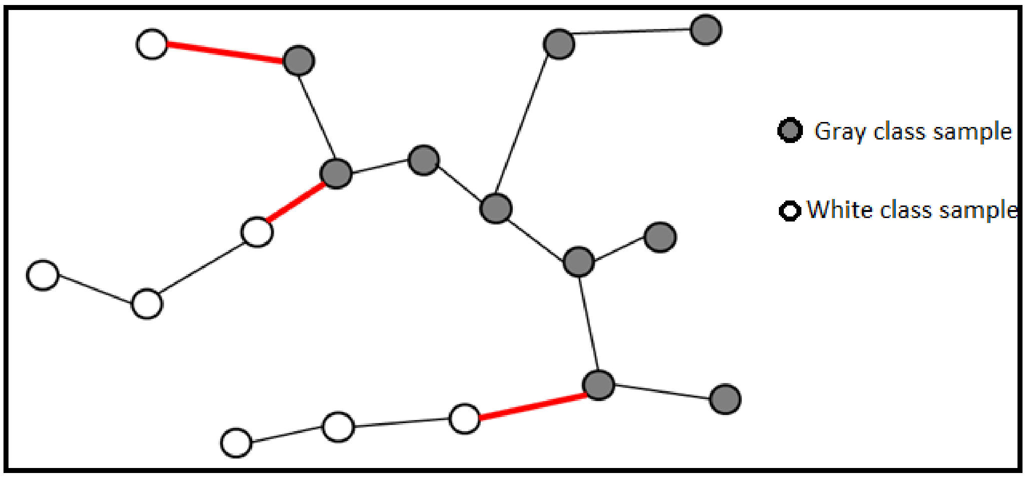


Figure 4.2: Example: Minimum Spanning Tree (MST)

Feature Efficiency (F3): The measure F3 computes the efficiency of individual features, which describes how much each feature contributes to class separation. Considering overlapping information, it calculates the number of samples lying outside of the overlapping region to the total number of samples. The formula for F3 is given by:

$$F3 = \prod_{i=1}^{d} \frac{\text{Number of samples outside overlapping region}}{\text{Total number of samples}}$$

where $i = 0, 1, .., d$ denotes the feature dimensions of the data.

The data is considered less complex when there are more samples lying outside the overlapping region. Thus, a greater value of F3 indicates lower complexity.

2. **Neighborhood Measures**: This category focuses on the data samples that are present in proximity belonging to different classes. There are three measures that provide information on nearest class data points.

   Fraction of Borderline Points (N1): The measure N1 considers distribution of samples among different classes in proximity. It computes the number of data samples belonging to a particular class residing next to a different class. This technique relies on developing a minimum spanning tree (MST) which connects neighboring data samples regardless of their class values. Figure 4.2 depicts an example of MST in which all data samples are connected to their nearest neighbors. Thicker red edges in the figure indicate the connections between opposite classes. The ratio of number of such thicker red edges to the total number of edges in the class yields measure N1. Therefore, lower value of N1 indicates lower complexity.

   Ratio of intra/inter class Nearest Neighbour distance (N2): Measure N2 computes the Euclidean distance between each data sample and its nearest neighboring sample, regardless of class information. Furthermore, the average intra-class distance (distance between samples belonging to the same class) is computed alongside the average inter-class distance (distance between samples belonging to opposite classes). The ratio of average intra-class to average inter-class distances gives measure N2. A lower value of N2 indicates lower complexity as the intra-class distance is smaller and the inter-class distance is greater.

Error rate of the Nearest Neighbour Classifier (N3): The presence of data samples of opposite classes in proximity affects the performance of classifier models. Thus, measure N3 computes the error rate of a nearest-neighbor classifier, estimated using the leave-one-out procedure. This measure is a good indicator of the separability of classes. As the measure takes error value into account, lower values of N3 indicate lower complexity.

3. **Dimensionality Measures**: Measures related to this category check for the importance of dimensonality of the data. Measures T1, T2 and T3 focus on information regarding the number of samples present per dimension and the number of samples present per PCA dimension. However, this information is redundant for this approach. The reason being this information cannot be used as optimization metric. Hence, only measure T4 is considered as the relevant complexity measure, whose description is given below.

   Ratio of the PCA dimension to the raw dimension (T4): Measure T4 considers the effect of the curse of dimensionality on the data. The method involves performing PCA to reduce the dimensionality of the data to the number of principal components containing 95% of original data variations. If data reduces to a lower number of dimensions, then data is said to be simple. On the other hand, data represented using many dimensions is complex. Measure T4 is the ratio of reduced PCA dimensions to the original dimensions. Therefore, a lower T4 value indicates that fewer dimensions are required to represent data samples and hence lesser complexity.

4. **Imbalance Degree (ID)**: The measure aims to compute the level of imbalance present in the data. Although the measure imbalance ratio [GSM12] is used to evaluate data imbalance between two classes, ID overcomes its drawback of generalizing it to multi-class problems [OIL17]. The imbalance ratio (IR) is a direct ratio of the number of samples in the majority class to the number of samples in the minority class. However, for multi-class problems, there exist more than one majority/minority class, thus making the ratio hard to indicate actual level of imbalance. Therefore, to tackle the issue, imbalance degree introduces three types of class distributions: empirical (θ), minority (l_m) and balanced (e). Empirical distribution refers to the overall class distribution of imbalanced data, whereas minority distribution considers only minority classes into account. The balanced distribution refers to the case in which all classes had an equal number of samples. ID then uses a distance metric to calculate the degree of divergence between the class distributions θ and l_m by comparing them with e. This number indicates how the empirical and minority distributions differ when compared to a balanced distribution. The formula to compute ID is given by:

$$ID = \frac{\delta(\theta, e)}{\delta(l_m, e)} + (m - 1),$$

   where δ is the Euclidean distance function, m is the number of minority classes. Lower value of ID corresponds to lower complexity.

Out of all the complexity measures described above, measures related to the feature overlapping category tend to address the complex data characteristics really well. Especially, the measure F1 addresses class overlapping characteristics, thereby keeping samples of the same class within a single partition/cluster. Hence, F1 measure is considered the primary complexity measure to deal with when partitioning the data.

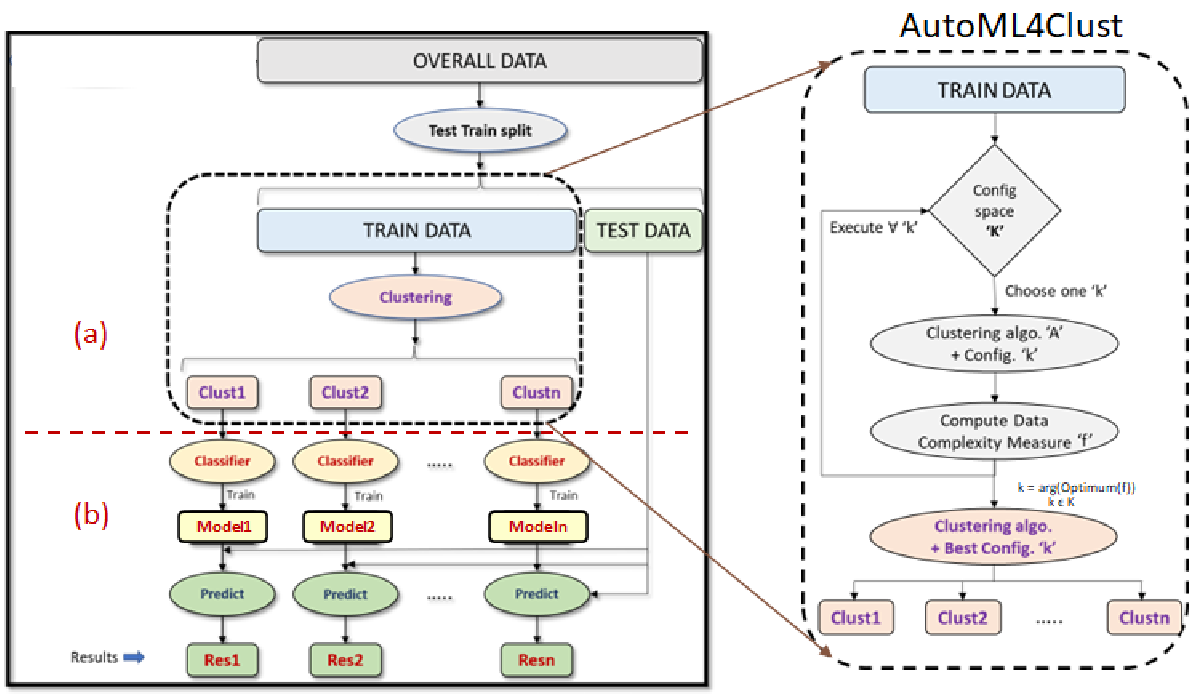


Figure 4.3: Hyper-parameter optimization of data-driven clustering using AutoML4Clust

4.3 Optimization of Data-driven Clustering and Data Complexity Measures

The goal of the proposed method is to partition the training data using data-driven clustering such that the complexity reduces. There exist numerous clustering techniques that cluster data based on various metrics. As a result, there is a need to experiment with different clustering methods for data partitioning, such as hierarchical clustering, distance-based clustering, and gaussian-based clustering. Furthermore, selecting and optimizing for the appropriate complexity measures becomes an obvious step in reducing data complexity. Therefore, the proposed method employs AutoML4Clust [TFS], which considers all possible types of clustering methods when partitioning the training data. After the data is clustered, the data complexity measure values of each partition are calculated. These values are averaged across all partitions to produce the final complexity value which is then used for optimization. Minimizing this final value yields the lowest complexity. Therefore, the best combination of clustering method and its hyper-parameter values that yields lowest complexity is estimated. Finally, the data is partitioned based on the best possible combination.

Figure 4.3 (a) illustrates the concept of using data-driven partitioning along with AutoML4Clust. The data is split into training and testing data for modeling and validation. The AutoML4Clust optimizer contains a configuration space (CS) with various clustering hyper-parameter values. The optimizer selects the configurations from the CS greedily. Only training data is considered, and the configuration chosen by the optimizer is applied. To this end, data is partitioned using a specific clustering method and its hyper-parameter values, and the value of the data complexity measure is computed for each partition. Various combinations of clustering methods and associated hyper-parameters are examined to determine the optimum value of the data complexity measure.

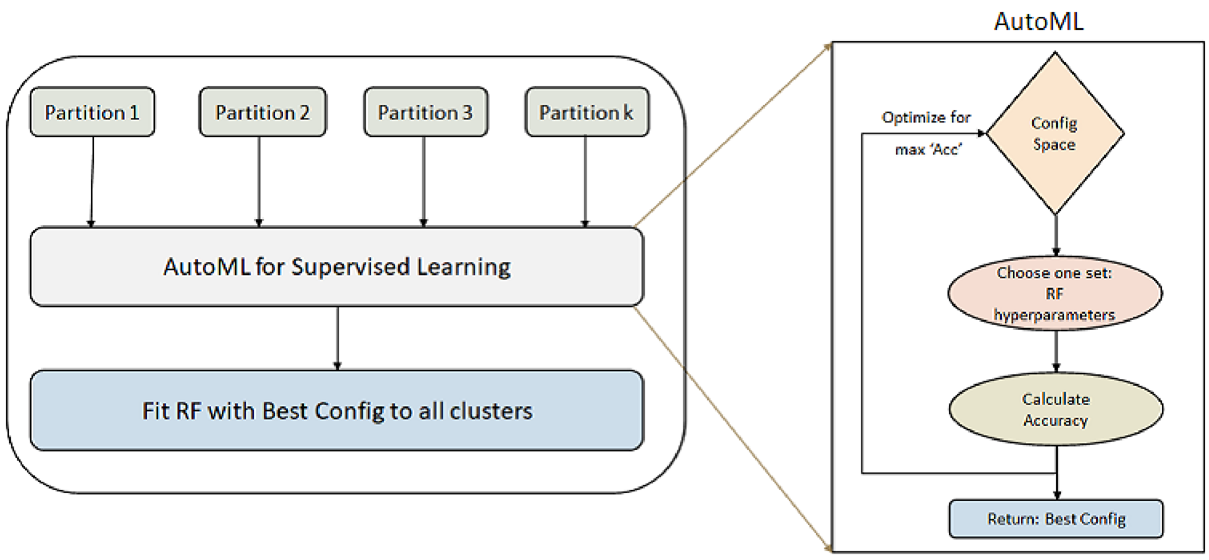


Figure 4.4: Hyper-parameter optimization of random forest on individual partitions using AutoML

The AutoML4Clust optimizer produces the combination that best partitions the training data, yielding the optimum complexity value across all partitions. Finally, the training data is partitioned according to the optimum configuration.

4.4 Classification using Random Forest

After partitioning the training data based on partition with the lowest complexity, a classification algorithm is modeled on each partition. A random forest classifier (RFC) is trained on each partition separately, using the samples and classes present in the partition. To this end, classifier models are created on individual partitions. RFC is employed as it constructs multiple decision trees by taking into account random samples and features. This bagging bootstrapping property prevents overfitting.

The Figure 4.3 (b) depicts the idea of applying random forest to partitioned data. The random forest classifier is modelled using the default hyper-parameter configuration. As a result, regardless of the number and type of samples, the same classifier is present on all partitions. This is done, however, to ensure consistency among classifiers across partitions. Because the baseline approach [HRM20] uses the same default settings, the results are compared to see how well the clustering-based classification technique performs when compared to a direct application of the classifier on the entire data set. The results are obtained by validating the classifier models with test data. These results here correspond to the values of accuracy and F1-score.

4.5 Hyper-parameter Optimization of Random Forest

This section focuses on how to optimize the hyper-parameters of the random forest on each partition. This is accomplished by employing the AutoML optimizer for supervised learning [WTMH21]. This optimizer contains a configuration space involving hyper-parameters of classifiers. It greedily chooses one configuration and computes accuracy of the system using test data present in the partition. AutoML experiments with various combinations of classification algorithm hyper-parameters and optimizes for the accuracy. This is used as optimization metric for AutoML. This procedure is performed on each partition separately. As a result, classifier present on each partition has different hyper-parameter settings in the end.

Figure 4.4 is regarded as an improved concept of the Figure 4.3 (b), with an extension being application of AutoML to classification stage as well. The training data is partitioned into 'k' partitions as specified in section 4.3. Further, AutoML trains each partition with a random forest classifier using hyper-parameters defined in its configuration space. The classifier then predicts the accuracy considering test data. The optimizer outputs the best possible configuration for which the accuracy of that partition is highest after considering various hyper-parameter configurations. As a result, overall accuracy improves across all partitions.

5 Prototypical Implementation

This chapter describes the prototypical implementation of the proposed concept in Chapter 4. Section 5.1 provides a general overview of the prototypical implementation. Section 5.2 provides an overview of the dataset that is generated using a synthetic data generator. This dataset is used for prototypical implementation. The implementation of the clustering algorithms is explained in Section 5.3. Section 5.4 covers the implementation of the Bayesian optimizer. Section 5.5 explains the usage of the API of the prototypical implementation.

5.1 General Overview of the Prototype

The prototypical implementation of the proposed concept can be found in a GitLab repository [1]. It uses Python as the programming language since many ML-libraries such as sklearn are available. The implementation contains four important packages. These are listed below:

- **Algorithm**: This package contains the implementations to execute the clustering algorithms. The algorithms K-Means [HW79], BIRCH [ZRL96] and GMM [Ras03] are imported from sklearn library [PVG+11], while the Bisecting K-Means package [SB01] is imported from a GitHub implementation [2].
- **Optimizer**: The implementation of the Bayesian optimizer used can be found in this package. This has been taken from the implementation available in GitHub [3].
- **Utils**: The python file *functions.py* contains all the implementations of utility functions that compute complexity measures described in Section 4.2. These functions are directly imported into the main code when computing the complexity.
- **Experiment**: All code related to the evaluation in Chapter 6 can be found in this package.

5.2 Dataset

The datasets are generated from a synthetic data generator tool, while its implementation is available in a GitLab repository [4]. It contains the module *DataGenerator.py* that generates data considering multi-class imbalance and heterogeneous characteristics. Further, the inputs such

[1]https://gitlab-as.informatik.uni-stuttgart.de/tschecds/master_thesis_tushar

[2]https://github.com/munikarmanish/kmeans

[3]https://github.com/fmfn/BayesianOptimization

[4]https://gitlab-as.informatik.uni-stuttgart.de/tschecds/imbalancedatagenerator

as the number of samples (N), the number of features (F), the number of classes (C), and the imbalance level can be specified. For this work, the following values are given to as inputs: $N = 1000, F = 100, C = 84, imbalancelevel = \text{`}medium'$.

```
1  import random
2  import numpy as np
3  from DataGenerator import ImbalanceGenerator
4  from Utility import train_test_splitting
5
6  # Set random seed for reproducing same results
7  np.random.seed(0)
8  random.seed(0)
9
10 # Parameters for Data generation
11 N = 1000
12 F = 100
13 C = 84
14
15 # Instantiate an imbalance generator
16 generator = ImbalanceGenerator(n_features=F, n=N, cls_imbalance="medium", c=C, features_remove_percent=0)
17
18 # Generate data
19 df = generator.generate_data_with_product_hierarchy()
20
21 # Split data into train and test
22 df_train, df_test = train_test_splitting(df, n_train_samples=int(0.75*N))
```

Listing 5.1: Example API of data generator

Listing 5.1 provides an example API of data generator. The required libraries are imported in lines 1-4. Lines 7-8 select the random seed so that the same data is reproduced during multiple runs. Lines 11-13 define parameters N, F and C required for data generation. In line 16, an instance of class *ImbalanceGenerator* is created that generates the dataset. It takes the parameters specified before, along with *imbalance level* being '*medium*'. This will generate data with normal imbalance characteristics. The argument *cls_imbalance* can take 5 different values, namely very imbalanced, imbalanced, medium, balanced and very balanced. The argument *features_remove_percent* specifies the percentage of features whose importance is made redundant during data generation. However, it is kept to zero because the objective of this work is to address other characteristics such as multi-class imbalance and heterogeneous data properties. Further, line 19 splits the data generated into training and testing. 75% of samples are considered for training while 25% are used for testing.

5.3 Clustering Algorithms

The implementations of the clustering algorithms are taken from scikit-learn [PVG+11]. These algorithms partition data into 'k' clusters, where the hyper-parameter 'k' is chosen by the optimizer. All the clustering algorithms have the hyper-parameter 'k' in common. Hence, K-Means, BIRCH and GMM implementations from sklearn are used, whereas Bisecting K-Means is extracted from an external GitHub source. K-Means algorithm is initialised using K-Means++ [AV06] technique for efficient results. To add a new clustering algorithm, the implementation of it needs to be defined. This implementation then needs to be added to the optimizer so that it is considered for data-driven partitioning. The file *Data_driven_partitioning_Optimized.py* needs to be adjusted in this regard.

5.4 Bayesian Optimizer

The implementation of Bayesian optimizer is taken from a GitLab repository [5]. A configuration space consisting of hyper-parameters and their bounds needs to be defined along with their data types. A black box function is defined by running a clustering algorithm over the data and evaluating the value of complexity measure with one of the metrics discussed in Section 4.2. Finally, the budget of the optimizer that corresponds to the number of iterations the optimizer needs to execute is also defined. To this end, built-in methods of *BayesianOptimization* class, namely *maximize()* or *minimize()* are called. These methods evaluate for the best black-box function result and thus provide the optimal hyper-parameter values. When applying the concept described in Chapter 4, exactly one optimizer is executed for each clustering algorithm and a chosen complexity metric. The clustering parameter 'k' is the only hyper-parameter to be optimized during data-driven partitioning. Parameter type of 'k' is specified as *integer*.

Further, during the optimization of hyper-parameters of random forest algorithm, various hyper-parameters of random forest such as number of estimators (*n_estimators*), maximum depth (*max_depth*), minimum samples per split (*min_samples_split*) are specified along with their ranges and data types. In this work, hyper-parameter ranges are given as: $n\_estimators = (10, 500), max\_depth = (1, 100), min\_samples\_split = (2, 10)$. Parameter types are specified as *integer*. However, the black-box function during this classifier optimization stage is the evaluation of accuracy value of the partitions. This is done by executing clustering, training random forest on each partition, and then evaluating accuracy of the partition in each iteration.

5.5 Usage of the API

This section presents the use of API of the prototypical implementation. The API code is executed in the PyCharm IDE that runs on a Windows 11 machine. However, it runs on different IDEs and also on different OS like Linux.

```python
from sklearn.cluster import KMeans
from sklearn.ensemble import RandomForestClassifier as RF
from utils import ft_F1, get_data, get_cluster_samples, predict_to_cluster
from bayes_opt import BayesianOptimization
from sklearn.metrics import accuracy_score, f1_score
import numpy as np

# Generate Dataset and perform train-test split
X_train, y_train, X_test, y_test = get_data()

# Cluster data using k-Means algorithm and for eac cluster, compute F1 complexity measure
def KMeans_F1(k, X_train, y_train):
```

[5]https://github.com/fmfn/BayesianOptimization

```
13     kmeans = KMeans(n_clusters=k, init="k-means++", n_init=10, max_iter= 300, random_state=42)
14     y_kmeans = kmeans.fit_predict(X_train)
15     F1 = []
16     for cluster in y_kmeans:
17         cluster_X_train, cluster_y_train = get_cluster_samples(X_train, y_train, cluster)
18         F1.append(ft_F1(cluster_X_train, cluster_y_train))
19     return np.mean(F1)
20 
21 # Create Bayesian optimizer with required parameters
22 optimizer = BayesianOptimization(f= KMeans_F1, pbounds={"k": (2, len(n) - 1),}, ptypes={'k': int})
23 
24 # Perform the optimizer loop until the budget is exhausted
25 optimizer.maximize(n_iter=10)
26 
27 # Find the optimum values of hyper-parameter
28 k_opt = optimizer.max['params']['k']
29 
30 # Cluster training data as per the optimum clustering parameter
31 kmeans = KMeans(n_clusters=k_opt, init="k-means++", n_init=10, max_iter=300, random_state=42)
32 y_clust = kmeans.fit_predict(X_train)
33 
34 # Compute accuracy and f1-score of each cluster
35 Accuracy = []
36 F1_score = []
37 for cluster in k_opt:
38     rf = RF(random_state=42)
39     cluster_X, cluster_y = get_cluster_samples(X_train, y_train, cluster)
40     rf.fit(cluster_X, cluster_y)
41     cluster_X_test, cluster_y_test = predict_to_cluster(X_test, X_train, y_train)
42     cluster_y_pred = rf.predict(cluster_X_test)
43     Accuracy.append(accuracy_score(cluster_y_test, cluster_y_pred))
44     F1_score.append(f1_score(cluster_y_test, cluster_y_pred, average='weighted'))
45 
46 # Display final results: Average of accuracy and F1 scores of all clusters
47 print(f'Results: Accuracy: {np.mean(Accuracy)} and F1-score: {np.mean(F1_score)}')
```

Listing 5.2: Example procedure for using the API

Listing 5.2 presents the adaptation of the API. It consists of a simple demonstration of the proposed concept using the K-Means algorithm for data-driven partitioning and the F1 measure as the optimization metric. It uses default parameters of random forest during classification stage. Initially, the required libraries are imported in lines 1-6. Line 9 executes the function *get_data()* that is responsible for generating the dataset with complex characteristics and then splitting it into train and test data. An example of generating such data can be found in Listing 5.1. Line 12 defines the *KMeans_F1* function, which partitions the training data into 'k' clusters. It then computes the F1 complexity measure on each cluster individually. The function *get_cluster_samples* aids in allocating samples to clusters. Finally, the function *KMeans_F1* returns the average F1 value across all clusters. Lines 13-19 illustrate this functionality.

In line 22, an instance of Bayesian optimizer is created that optimizes the black-box function *KMeans_F1*. Further, the configuration space consists of the hyper-parameter 'k' with its value range and its data type. The range of 'k' is from 2 to (c-1), where 'c' is the number of classes in the dataset. Line 25 performs the execution of the optimizer to maximize the F1 measure over all clusters. It is executed until the budget (*n_iter*) is exhausted. In line 28, the results of the optimizer are extracted that gives the optimum value ’k_opt’. Once the optimal clustering parameter ’k_opt’ is determined, the training data is partitioned into ’k_opt’ clusters using K-Means. Lines 31-32 demonstrate this operation.

After the clusters are formed, line 37 considers each cluster individually, and a random forest classifier with default parameter settings is trained on data samples belonging to the cluster (lines 38-40). Further, in lines 41-42, test data belonging to the cluster is predicted (*cluster_y_pred*). In lines 43-44, these predictions are used to evaluate the values of accuracy and F1-score. This is done by comparing the predicted values of *cluster_y_pred* with the truth labels *cluster_y_test*.

Line 44 displays the final results which is the average of accuracy and F1-scores over all the clusters.

6 Evaluation

In this chapter, a comprehensive evaluation of the proposed concept is performed. In Chapter 5, a prototypical implementation of the proposed concept is demonstrated. However, there exist various clustering methods and hyper-parameter configurations that produce different results. Therefore, various experiments are conducted using various datasets and complexity measures, each of which has an effect on the results. Information regarding the experimental setup is described in section 6.1. Section 6.2 discusses various optimization metrics that aid for efficient data partitioning. Section 6.3 discusses the results obtained by applying the proposed method to a specific dataset that resembles a real-world use case. Section 6.4 shows the comparison of results to existing state-of-the-art approaches. Various trials conducted for different data characteristics are discussed in section 6.5.

6.1 Experimental Setup

This section focuses on the design of the conducted experiments. It explains the used hardware and software. Also, the setup of the optimizer used in implementing AutoML4Clust is presented. Further, the procedure used in accuracy evaluation of the proposed system using test data is described.

6.1.1 Hardware and Software

The experiments are performed on a Windows machine that operates on Windows 11. It consists of a 6-core CPU with 2.5 GHz and 16 GB RAM. The implementation is based on Python. A virtual python environment with Python 3.7.4 version is used. PyCharm IDE with version 2020.2.3 is used to develop the implementation and then analyze the results. The whole prototype is provided in the Gitlab repository. The required libraries and versions are listed in the *requirements.txt* [1] file in the same GitLab repository.

6.1.2 AutoML4Clust Implementation

The optimizer AutoML4Clust used for evaluation is based on the Bayesian model. Bayesian optimizer needs several parameters to be defined, such as objective function, parameter bounds, parameter type, and the budget (number of iterations). Hyper-parameters of several clustering algorithms are defined along with their predefined bounds that make up the configuration space. Each clustering algorithm is executed separately, and the optimizer chooses a value of hyper-parameter ‘k’

[1] https://gitlab-as.informatik.uni-stuttgart.de/tschecds/master_thesis_tushar/-/blob/main/requirements.txt

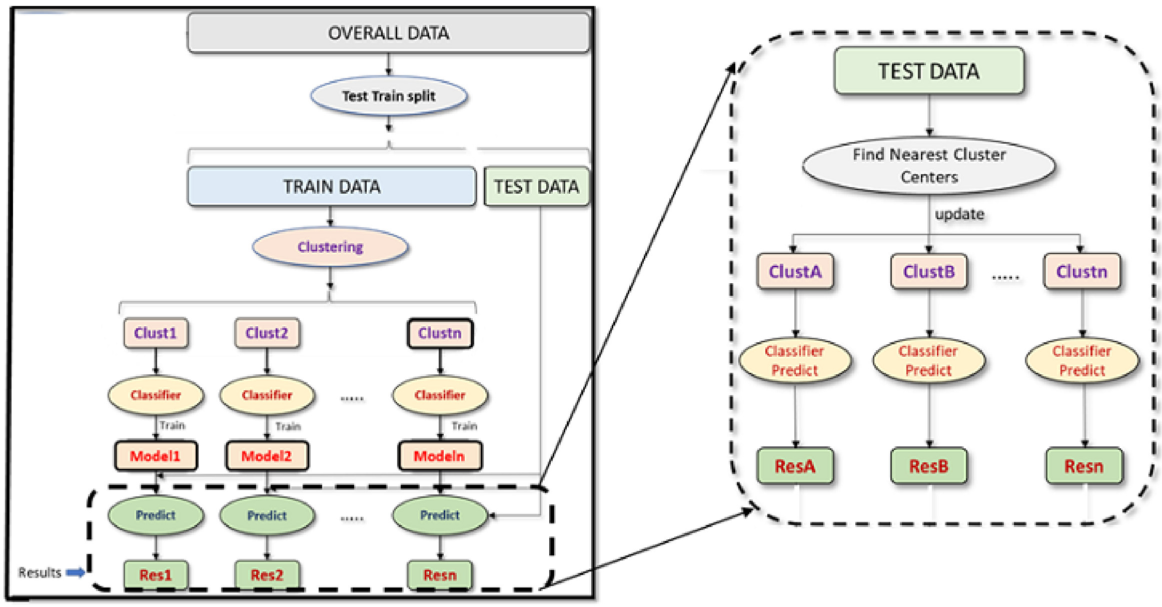


Figure 6.1: Data validation procedure

greedily, and clusters the data into 'k' chosen partitions. Further, it computes a metric (clustering or complexity metric) averaging over all partitions. It then optimizes the value of 'k' for the lowest complexity. The range of selecting hyper-parameter is given by k = 2,. . . (c-1), where c is the number of classes. Therefore, the total number of instantiations employed by the optimizer correspond to the number of clustering algorithms times the number of complexity measures.

The optimizer executes for 15 iterations to find the optimum hyper-parameter value. Other parameters corresponding to clustering algorithms are set to default scikit-learn parameters.

6.1.3 Implementation

The AutoML4Clust optimizer executes each of the four clustering algorithms (K-Means, BIRCH, GMM, and Bisecting K-Means) multiple times, and the training data is partitioned into 'k' partitions (as explained in Section 4.3). Then, on each partition, a random forest classifier with the default parameter settings is trained. Thus, a classifier model is present on each partition.

6.1.4 Validation using Test-Data

This subsection emphasizes on evaluation of accuracy. A performance metric like accuracy aids in estimating the goodness of the trained classification model. As stated in the Section 4.5, accuracy is obtained by validating test data against trained classifier models. This is done by considering each sample of test data and the cluster (partition) information. As a result, assigning test samples to respective partitions is crucial when determining model accuracy.

Considering Figure 6.1, the left side is the procedure dealing with partitioning of training data using AutoML4Clust. However, during the prediction stage, each test sample is assigned to the partition whose center is closest to it. Once the test data has been assigned to all partitions, the performance metrics accuracy and F1-score are determined.

6.2 Investigation of different Optimization Metrics

This section deals investigates the use of various optimization metrics that measure the complexity of partitions during data-driven clustering. The procedure involves application of AutoML4Clust considering the value of complexity measure as optimization metric and providing the optimum data partitioning of the training data. Further, classifiers are trained on each partition, and performance metrics accuracy and F1-score are calculated considering test data. Following sub-sections deal with various optimization metrics employed in this work.

6.2.1 Considering individual Data Complexity Measures

Data complexity measure focuses on a specific pattern in the data to help quantify the complexity for classification. However, numerous complexity measures evaluate specific data patterns. This experiment validates the results of all eight complexity measures discussed in Section 4.2. One complexity measure is considered at a time and chosen to be the optimization metric for AutoML4Clust. Data is partitioned for optimum value of this complexity measure. A classifier is then trained on each partition and validated against test data. For each complexity measure, the performance metrics accuracy and F1-score are recorded.

The Figure 6.2 shows the results obtained through implementing various complexity measures. F1 measure from the class overlapping category performs the best. The reason being this measure addresses heterogeneous characteristics, thus distinguishing overlapping classes better compared to other category measures. It even has influence on imbalanced data, as minority classes get separated into different partitions. Neighbourhood measures N1, N2 and N3 focus on separating out the samples belonging to different classes in distance. But due to the complex characteristics present in data, these category measures provide average results. Measure T4 uses PCA to represent data in lower dimensions and compares the number of dimensions to the original data. Due to the fact that complex characteristics sometime still persist in lower dimensions, this measure also performs moderately. The measure Imbalance Degree (ID) focuses on reducing data imbalance. However, it fails to provide better results due to the presence of other complex data characteristics.

6.2.2 Combining the best-performing Complexity Measures

The experiment aims to combine two or more complexity measures to examine whether the combination solves the problem of multiple complex characteristics present in the data. As a result, the best-performing complexity measures are considered from Table 6.4, and their combinations are evaluated. Performance metrics are recorded for each pair of combination.

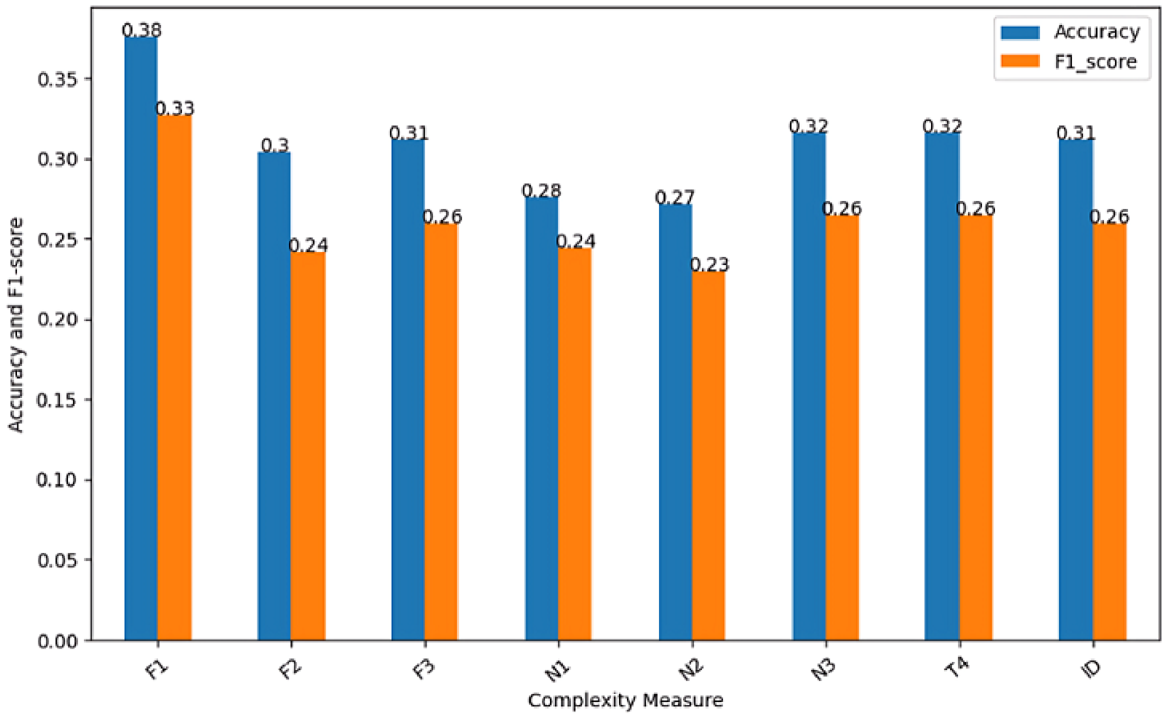


Figure 6.2: Accuracy results for using AutoML4Clust with the different complexity measures

The Figure 6.3 shows the accuracy and F1-score values achieved for each combination of complexity measures. Top-performing measures are combined and optimized for their mean values. Although these measures perform better in isolation, that they do not work well together. For example, when the measures F1 and F2 are combined, the results are poor because some partitions may have lower F1 and higher F2 values, or vice versa. As a result, the partitions may end up with complex characteristics related to either of the measures. The same behavior is observed when class overlapping and imbalance degree measures (F1+ID) are combined. Although the highest achievable accuracy is around 28% (combining measures F1, F2 and F3), it is still poor when compared to using individual complexity measures. As a result, the combination of complexity measures does not work well with given complex data characteristics.

6.2.3 Comparison to using Clustering Metrics

Clustering metrics evaluate clustering results by considering the similarity of samples within clusters. As a result, these metrics can be used instead of complexity measures, and their values can be optimized for better data partitioning. Thus, the experiment deals with employing the clustering metrics discussed in section 2.1.2. Following the optimization of the metrics for data partitioning, the usual process of classifier modeling is performed. The outcomes are shown in Table 6.1.

Referring to the results, although clustering metrics generally evaluate clusters based on similarity, they do not consider the class labels within each cluster (partition). Rather, the metrics focus on how samples from different clusters are separated from one another. But, in the presence of complex

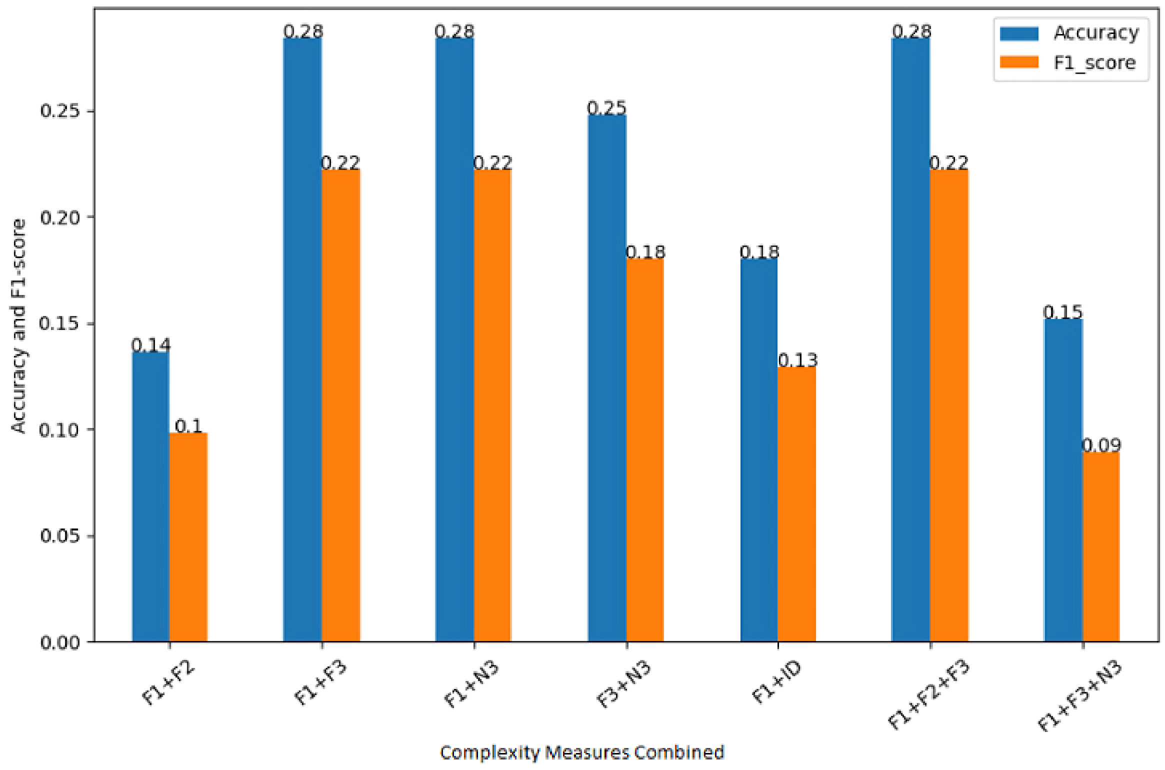


Figure 6.3: Accuracy results for using AutoML4Clust with combined complexity measures

Cluster Metric	Optimum-k	Accuracy	F1-score
Davies-Bouldin Index (DBI)	2	0.30	0.23
Calinski-Harabasz Index (CH)	2	0.30	0.23
Silhoutte score (SIL)	2	0.30	0.23
Proposed Approach (K-Means + F1 measure)	48	0.376	0.327

Table 6.1: Performance of clustering metrics in data-driven partitioning

data characteristics, this type of evaluation usually fails as there is a need to evaluate partitions based on class labels. The results are surprisingly same for all the three clustering metrics employed. It is because all these metrics focus on evaluating cluster separation. As a result, using clustering metrics to address complex data characteristics does not lead to improved results.

6.3 Data-driven Partitioning Results

This section discusses the results obtained for the proposed approach. It also describes about the partitions formed during clustering. Further, an experiment examines the highest achievable accuracy on the data in focus. The data with complex characteristics as described in Section 5.2 is considered and analyzed. The working environment for implementation and optimizer parameters are configured as described in section 6.1. The implementation of proposed approach is carried out considering different clustering algorithms and Fisher's Discriminant Ratio (F1) as the complexity measure. The proposed method finds the best partitioning scheme by applying an optimizer for each clustering method separately. Table 6.2 presents the results.

Clustering Method	**Optimum k**	**Accuracy**	**F1-score**
K-Means	48	0.376	0.327
BIRCH	67	0.292	0.226
GMM	69	0.288	0.253
Bisecting K-Means	22	0.272	0.231

Table 6.2: Results of data-driven approach

K-Means clustering outperforms other clustering methods in terms of accuracy and F1-score. The table also provides information on the optimum 'k' value chosen by the optimizer, along with performance metrics.

6.3.1 Accuracy

Accuracy is a vital factor that decides the performance of the proposed clustering-based classification technique, as it evaluates the number of times the system has performed correctly. In the case of the proposed method, K-Means, the distance-based clustering approach, tends to perform better as it partitions the training data into k=48 partitions based on selected complexity measure value. Accuracy of 37.6% is observed considering the complex dataset. On the other hand, clustering algorithms BIRCH and GMM tend to over-estimate the number of clusters (partitions). Bisecting K-Means splits dataset into fewer partitions yet providing average results. This behavior is usually observed in divisive-based hierarchical clustering algorithms, as they face difficulty handling different sized clusters. If a partition contains adequate number of samples for each class, the model is well trained. As a result, the number of partitions, classes and the samples within each partition influence the accuracy.

6.3.2 F1-score

Accuracy does not deal with majority/minority class separately. It often occurs that a classifier might predict all samples to a majority class and still achieve high values for accuracy. Hence, metrics like precision and recall need to be computed that quantify the relevant results retrieved in terms of majority and minority classes separately. F1-score considers precision and recall values that depict the behavior of the classifier towards both majority and minority classes. As shown in the Table 6.2, K-Means with Fisher's Discriminant Ratio performs better in the case of F1-score. Although other clustering methods over-estimate the number of clusters while partitioning and hence perform satisfactorily. They provide F1-score revolving around 25%, while K-Means dominates with 33%.

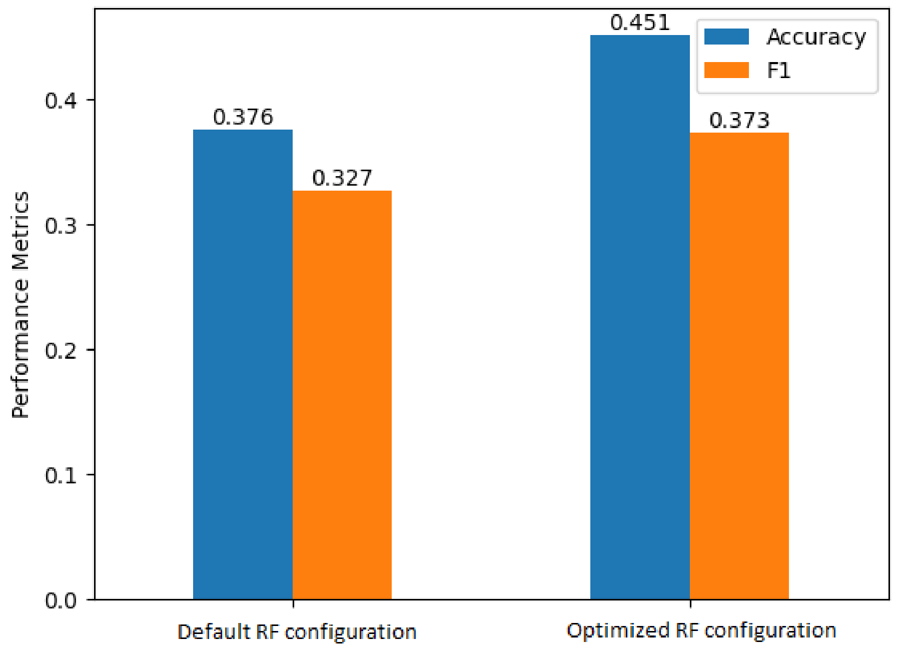


Figure 6.4: Performance metrics comparison of optimized and default RF classifiers

6.3.3 Results of Optimized Classifiers

This subsection focuses on the outcomes of the hyper-parameter optimization of the random forest classifier discussed in Section 4.5. This is carried out to further improve the performance of classifiers on individual partitions. The training data is partitioned using AutoML4Clust as per the optimum complexity measure values as demonstrated in Section 4.3. Thereafter, AutoML for supervised learning is applied to optimize hyper-parameters on a random forest classifier to achieve maximum accuracy considering test data. The Figure 6.4 illustrates the increase in accuracy and F1-score values for optimized classifiers when compared to the classifiers with default configuration setting. As the configuration of random forest classifier (RFC) is optimized for highest accuracy value on each partition, the results are higher compared to the default RFC.

6.3.4 Information on Partitions formed

Figure 6.5 depicts the average samples and classes present in each partition for different clustering methods employed. This information is crucial to understand how a clustering algorithm has partitioned the data and to understand the subsequent accuracy results. A sufficient number of samples in the partition indicates that classification algorithms are well trained, whereas a larger number of classes in the partition hinders a classifier's performance. Insufficient number of samples in partition also degrades classifier's performance. For example, Bisecting K-Means has around 17 classes per partition, which makes training a classifier difficult. Methods BIRCH and GMM, on the other hand, have only about 11 samples, which are insufficient for learning. Comparably, K-Means has a better ratio of samples to clusters, and hence stands out to be better in terms of accuracy and F1-score.

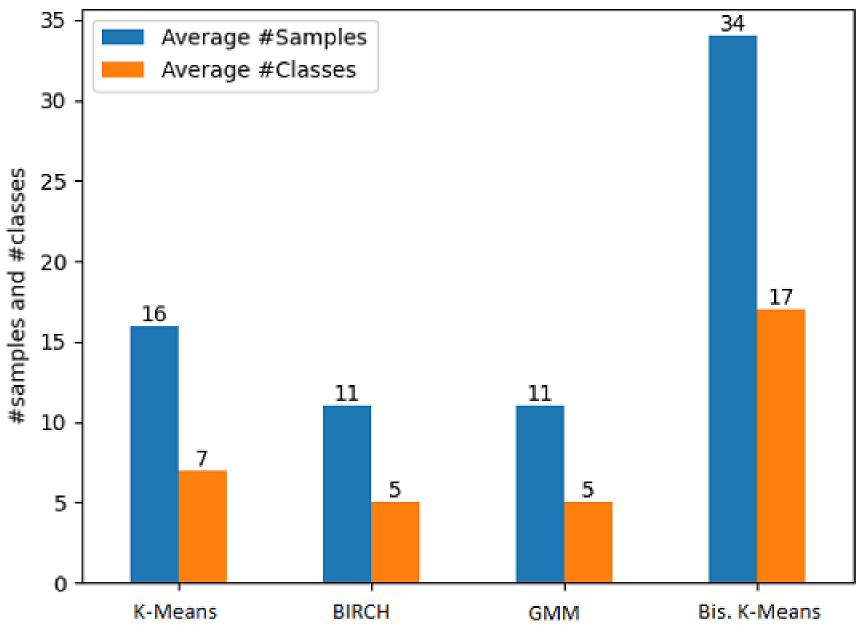


Figure 6.5: Average samples and classes per partition

6.3.5 Calculation of the upper bound of best achievable accuracy

An experiment is carried out to determine the maximum accuracy achievable on the dataset in question. Instead of maximizing complexity during clustering, the model maximizes accuracy while using the same training and testing data. The Figure 6.6 shows the accuracy values that can be achieved on the dataset using the proposed approach. The K-Means algorithm achieves the highest accuracy of 37.6%, which is also achieved by the proposed method. As a result, optimizing K-Means with the F1 complexity measure for partitioning data performs proficiently on data with complex characteristics.

6.4 Comparison to existing State-of-the-art Approaches

This section discusses the results obtained by evaluating the related work techniques presented in chapter 3. The overall experiment is carried out for different data characteristics. The performance metrics of the techniques for these data variations are analysed.

6.4.1 Dataset Generation

When comparing several techniques, multiple datasets provide better proof for validation. As a result, multiple datasets are created in addition to the dataset discussed in Section 5.2. As the data generator tool works randomly, each dataset generated is different from another. Therefore, in

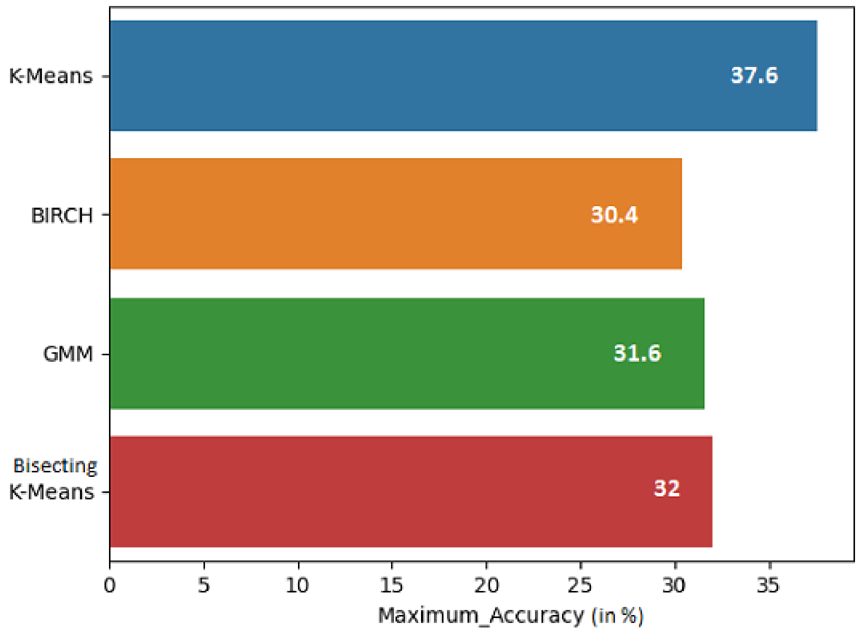


Figure 6.6: Upper bounds of maximum achievable accuracy

addition to the dataset in question (1000 samples, 100 features, 84 classes), additional datasets are generated by varying the number of samples (N), the number of features (F) and the number of classes (C).

6.4.2 The Experiment

The experiment uses previously discussed multiple datasets to implement the related works. The first technique involves the implementation and execution of SMOTE technique that addresses data imbalance. SMOTE is executed on training data to synthetically add samples to minority classes such that the class distribution is balanced. To this end, a random forest classifier with default parameter settings is trained, and the model is validated against test data. The second technique involves the application of random forest, an ensemble method, directly over the whole training data. The third technique implements the concept proposed by Hirsch et al. [HRM20], which involves SPH to partition the training data, followed by CPI to reduce data imbalance. A random forest classifier is modeled on each partition, and the results are captured. Finally, the proposed method of data-driven partitioning (using K-Means clustering and Fisher's Discriminant Ratio (FDR) measure as optimization metric) is employed, and the performance metrics are recorded.

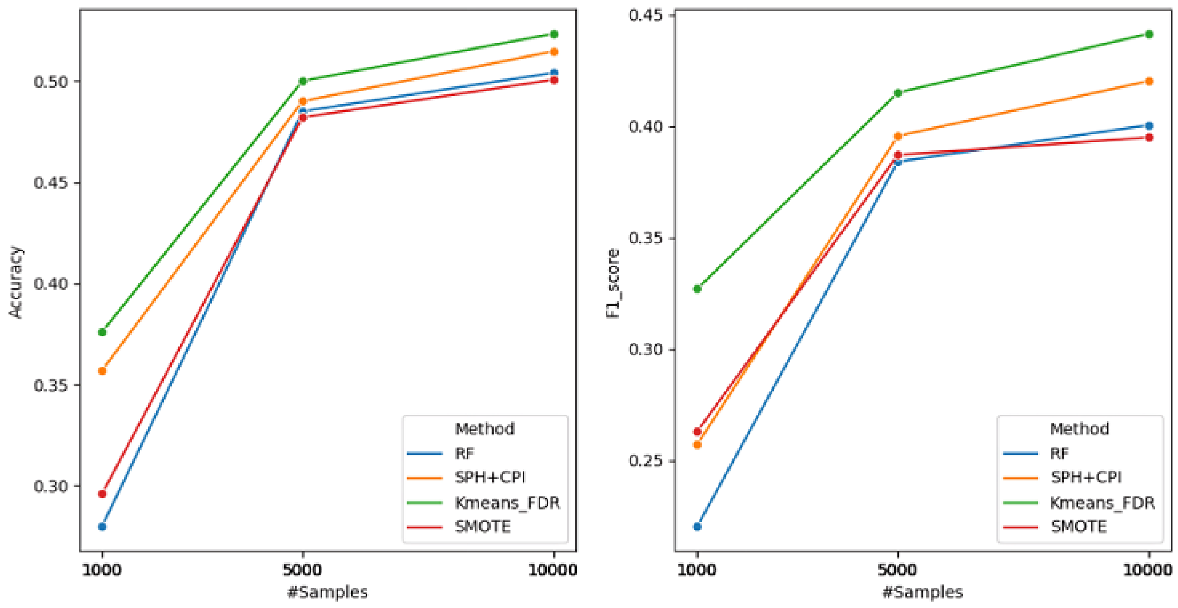


Figure 6.7: Accuracy and F1-scores of baseline approaches for varying samples

6.4.3 Results

Figures 6.7, 6.8 and 6.9 show the performance of various techniques discussed over the multiple datasets for varying number of samples, features and classes. Accuracy values of the approaches are recorded and plotted in the figure. In general, directly applying the random forest over training data gives poor results due to the complex data characteristics. Further, SMOTE technique also performs comparably with the previous technique and provides inferior results, as it focuses only on data imbalance. The approach by Hirsch et al. using SPH and CPI method performs well compared to the previous two techniques, as it focuses on both data imbalance and heterogeneous characteristics of data. Nevertheless, the proposed data-driven partitioning approach using K-Means clustering and Fisher's Discriminant Ratio (FDR) measure provides better accuracy in all three datasets. In general, for all techniques, accuracy can be seen increasing due to the increased number of training samples as well as decreasing number of classes. However, for the varying features, due to the effect of bias-variance, the lower and higher values of features produce better results.

In the case of the F1-score, considering the Figure 6.7, baseline approaches perform almost identically for datasets with an average number of samples (N=5000). The behavior for other characteristics is similar as of accuracy. However, the data-driven partitioning approach outperforms others significantly across all datasets. The reason being the reduction of complexity by partitioning the data as per the optimum complexity measure value. This approach results in producing partitions that contain samples of similar characteristics that can be trained efficiently by a classifier. Further, majority and minority classes are present in different partitions thus enhancing the overall performance.

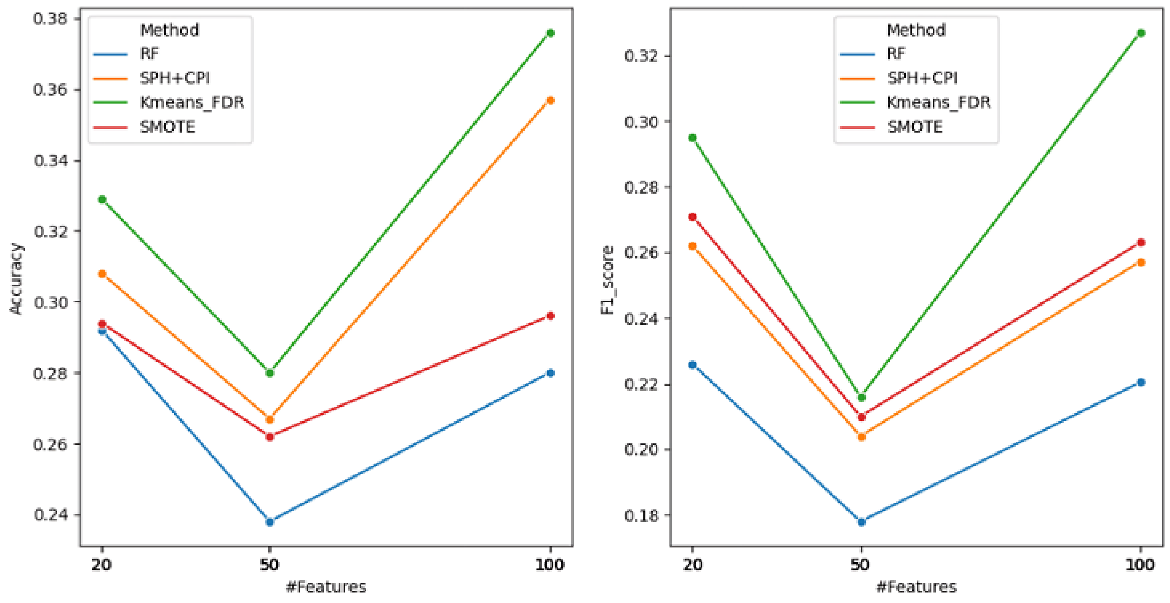


Figure 6.8: Accuracy and F1-scores of baseline approaches for varying features

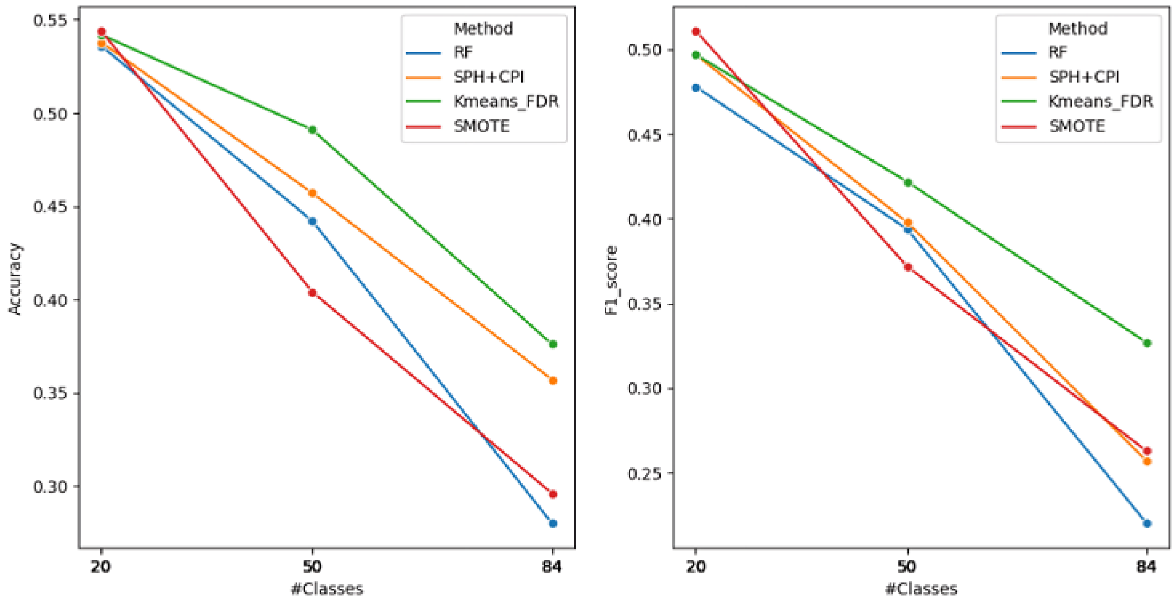


Figure 6.9: Accuracy and F1-scores of baseline approaches for varying classes

6.4.4 Performance of Majority-Minority classes

As the data deals with multi-class imbalance characteristic, it becomes crucial to find how majority and minority classes perform. As a result, the experiment computes the accuracy of each class and assigns it either majority or minority set based on the number of samples present in that class. The mean accuracy of the majority and minority groups is then calculated, which represents the overall system performance.

Class	Random Forest	SMOTE	SPH+CPI	Data-driven Partitioning
Majority Class	0.276	0.22	0.29	0.32
Minority Class	0	0.07	0.05	0.06

Table 6.3: Majority and minority class accuracy

The Table 6.3 shows the results of accuracy values achieved by the majority and minority classes. The random forest baseline performs poorly with minority classes, as the majority classes have a significant influence on classifier training. It also performs worse with majority classes compared to data-driven partitioning approach. Further, SMOTE baseline approach deals with minority classes very well, although its overall accuracy is low. The approach involving SPH and CPI also performs better with minority classes, but not the best. However, the data-driven partitioning approach not only outperforms SMOTE and SPH in terms of majority class accuracy, but also provides comparable results with SMOTE in terms of minority classes.

6.5 Comparing Results of Validation Datasets

This section examines the performance of the proposed data-driven technique on a variety of validation datasets employed. It also goes over how the model responds to changes in data characteristics.

6.5.1 Preparation of Datasets

Datasets with varying data characteristics are developed to evaluate the proposed method. Analysing performance metrics for these datasets show whether the concept is robust to any complex data characteristics. Hence, apart from the specific dataset used before, numerous other datasets are generated using a data generator model that almost imitate real-world data. Variations are performed on three key parameters that comprise the dataset: the number of samples (**N**), the number of features (**F**), and the number of classes (**C**). Two parameters are fixed in each trial, while the third is changed. As a result, additional datasets are generated that are used to evaluate the model. Apart from that, because the default case has the most complex data characteristics, two more datasets are generated to represent the simplest and moderate complexity cases, and the results are recorded.

Comment	N	F	C	Accuracy	F1-score
Default case: Most Complex Scenario	1000	100	84	0.376	0.327
Variation in N	5000	100	84	0.5	0.415
	10000	100	84	0.523	0.441
Variation in F	1000	20	84	0.329	0.295
	1000	50	84	0.28	0.216
	1000	80	84	0.267	0.208
Variation in C	1000	100	20	0.542	0.497
	1000	100	50	0.491	0.422
Least Complex Scenario	10000	20	20	0.67	0.615
Moderate Complex Scenario	5000	50	50	0.593	0.518

Table 6.4: Validation results

6.5.2 Results of Validation Datasets

The Table 6.4 shows the accuracy and F1-score values for various validation datasets. Initially, the default dataset with the most complex data characteristics is considered. Second set considers variation in number of samples. As the number of samples increase, much data gets into training and testing the model. As a result, the performance of model increases. Variation in the number of features, on the other hand, is an interesting behavior to observe. It is due to the concept of bias-variance trade-off. Typically, in the presence of more features, model is set to incur more variance, whereas fewer number of features would in turn result in high bias. However, it is dependent on the features that represent the complex data characteristics. As the number of features increases in this case, the accuracy values tend to decrease until F=80. Further, variations in number of classes goes as expected. With increase in number of classes, the performance metric values go down. This is because the number of samples are kept same while the number of classes increase, which results in fewer samples per class.

Furthermore, the model is expected to perform better for two datasets that are less complex than the standard case. Referring to the last two records from Table 6.4, as intended, the model produces very good results in this case. The highest accuracy and F1 score are obtained for the least complex dataset, as it has sufficient data samples and fewer classes. This behavior of the proposed approach proves that it is robust to various complex data characteristics.

7 Conclusion and Outlook

Due to the massive amount of data generated worldwide, it becomes crucial to perform data analysis to gain more insights into the data. The process includes applying data mining techniques using machine learning methods to extract information from the data. Since the real-world data typically consists of various complex characteristics such as multi-class imbalance and heterogeneous features, applying machine learning algorithms to such data provides poor results. Current research shows that partitioning the data can reduce the effects of complex characteristics and improve the accuracy. They use domain knowledge to partition the data so that similar data samples lie in the same partition. However, these systems depend upon the domain knowledge that is not always available in a real-world scenario. As a result, reducing the complexity of complex data characteristics becomes challenging.

In this work, an approach was developed to partition the training data using data-driven clustering to reduce the complexity. To this end, various measures that measure the complexity of the partitions were investigated. AutoML4Clust, an AutoML concept for unsupervised learning method was employed to provide the best results for various clustering algorithms along with their hyper-parameters. The proposed approach uses clustering to partition training data with complex characteristics such that the overall complexity of partitions is minimized. Following the formation of the partitions, random forest classifiers were modelled on each partition. The hyper-parameters of classifiers were further optimized using AutoML for supervised learning method to achieve maximum accuracy. A sample from test data was assigned to the cluster that had its center nearest to the sample.

The proposed approach was prototypically implemented, and the implementation was used for a comprehensive evaluation. To this end, 4 clustering algorithms were employed to cluster the training data. 8 complexity measures, and their combinations were used as optimization metrics to measure the complexity of the clustering results. Furthermore, the values of these metrics were optimized to achieve the lowest average complexity across all partitions. The experiments revealed that using K-Means clustering and optimizing for Fisher's Discriminant Ratio (F1) measure provided the best results in reducing the complexity and thereby providing better accuracy values. Other clustering algorithms overestimated the number of partitions and thus produced average accuracy results. Further, the proposed approach was compared with various state-of-the-art approaches over 9 validation datasets involving different data characteristics. The results proved that the proposed approach outperformed all other techniques in terms of accuracy and F1-score. The outcomes also revealed that the algorithms that predict majority and minority classes performed better than the other approaches. Additionally, the experiments also revealed that by using K-Means clustering and optimizing for the F1 measure to partition the data produced accuracy that was equal to the same value when optimizing for the accuracy directly. This demonstrates that the proposed approach successfully reduced the complexity of the training data, and that the classifiers' hyper-parameters were optimized for better performance.

Considering the dataset with complex characteristics, the results show that the proposed approach's accuracy value increased by 30% when compared to the approach that directly applies the classifiers to the entire training data. Furthermore, when the proposed approach was compared to the technique that used domain knowledge, it was discovered that there was a 5% increase in accuracy and F1-score. This shows that the data partitioning using clustering was able to reduce the complexity of training data.

This work presented an approach for data partitioning that used various clustering algorithms, but applied them individually. However, multiple clustering algorithms can be combined to produce results that consider the outcomes of all clustering methods into account [BO04] [FJ05]. Additionally, the proposed concept could be examined as to whether it is also able to address different data characteristics. It can be investigated how complexity measures can aid in data partitioning while reducing the complexity associated with such data.

This work employs a random forest classifier on each partition by optimizing its hyper-parameters to achieve high accuracy. However, other ensemble methods [Rok05] [Rok10] could be examined by modelling them on each partition and further checked for performance metrics such as accuracy and F1-score. Furthermore, different optimizers [FH19] may be used for hyper-parameter optimization in AutoML systems, and their performance could be evaluated.

Bibliography

[AV06] D. Arthur, S. Vassilvitskii. *k-means++: The advantages of careful seeding*. Tech. rep. Stanford, 2006 (cit. on pp. 18, 42).

[BB12] J. Bergstra, Y. Bengio. "Random search for hyper-parameter optimization." In: *Journal of machine learning research* 13.2 (2012) (cit. on p. 22).

[Bij13] L. Bijuraj. "Clustering and its Applications". In: *Proceedings of National Conference on New Horizons in IT-NCNHIT*. Vol. 169. 2013, p. 172 (cit. on p. 17).

[BO04] C. Boulis, M. Ostendorf. "Combining multiple clustering systems". In: *European conference on principles of data mining and knowledge discovery*. Springer. 2004, pp. 63–74 (cit. on p. 62).

[Bot18] A. Botchkarev. "Performance metrics (error measures) in machine learning regression, forecasting and prognostics: Properties and typology". In: *arXiv preprint arXiv:1809.03006* (2018) (cit. on p. 23).

[Bre96] L. Breiman. "Bagging predictors". In: *Machine learning* 24.2 (1996), pp. 123–140 (cit. on pp. 21, 28).

[BS16] G. Biau, E. Scornet. "A random forest guided tour". In: *Test* 25.2 (2016), pp. 197–227 (cit. on p. 21).

[CBHK02] N. V. Chawla, K. W. Bowyer, L. O. Hall, W. P. Kegelmeyer. "SMOTE: synthetic minority over-sampling technique". In: *Journal of artificial intelligence research* 16 (2002), pp. 321–357 (cit. on p. 27).

[CH74] T. Caliński, J. Harabasz. "A dendrite method for cluster analysis". In: *Communications in Statistics-theory and Methods* 3.1 (1974), pp. 1–27 (cit. on p. 21).

[DB79] D. L. Davies, D. W. Bouldin. "A cluster separation measure". In: *IEEE transactions on pattern analysis and machine intelligence* 2 (1979), pp. 224–227 (cit. on pp. 17, 21).

[DZLL21] G. Du, J. Zhang, S. Li, C. Li. "Learning from class-imbalance and heterogeneous data for 30-day hospital readmission". In: *Neurocomputing* 420 (2021), pp. 27–35 (cit. on p. 13).

[EK21] B. J. Erickson, F. Kitamura. "Magician's corner: 9. Performance metrics for machine learning models". In: *Radiology: Artificial Intelligence* 3.3 (2021) (cit. on p. 23).

[FH19] M. Feurer, F. Hutter. "Hyperparameter optimization". In: *Automated machine learning*. Springer, Cham, 2019, pp. 3–33 (cit. on pp. 22, 62).

[FHR18] W. Feng, W. Huang, J. Ren. "Class imbalance ensemble learning based on the margin theory". In: *Applied Sciences* 8.5 (2018), p. 815 (cit. on p. 28).

[FJ05] A. L. Fred, A. K. Jain. “Combining multiple clusterings using evidence accumulation”. In: *IEEE transactions on pattern analysis and machine intelligence* 27.6 (2005), pp. 835–850 (cit. on p. 62).

[FLG+13] A. Fernández, V. LóPez, M. Galar, M. J. Del Jesus, F. Herrera. “Analysing the classification of imbalanced data-sets with multiple classes: Binarization techniques and ad-hoc approaches”. In: *Knowledge-based systems* 42 (2013), pp. 97–110 (cit. on p. 27).

[FTR14] L. G. Fahad, S. F. Tahir, M. Rajarajan. “Activity recognition in smart homes using clustering based classification”. In: *2014 22nd International Conference on Pattern Recognition*. IEEE. 2014, pp. 1348–1353 (cit. on p. 29).

[Gan12] V. Ganganwar. “An overview of classification algorithms for imbalanced datasets”. In: *International Journal of Emerging Technology and Advanced Engineering* 2.4 (2012), pp. 42–47 (cit. on p. 13).

[GBV20] M. Grandini, E. Bagli, G. Visani. “Metrics for multi-class classification: an overview”. In: *arXiv preprint arXiv:2008.05756* (2020) (cit. on p. 23).

[GFB+11] M. Galar, A. Fernandez, E. Barrenechea, H. Bustince, F. Herrera. “A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42.4 (2011), pp. 463–484 (cit. on p. 27).

[Gra04] Y. Grandvalet. “Bagging equalizes influence”. In: *Machine Learning* 55.3 (2004), pp. 251–270 (cit. on p. 21).

[GSM12] V. Garcıa, J. S. Sánchez, R. A. Mollineda. “On the effectiveness of preprocessing methods when dealing with different levels of class imbalance”. In: *Knowledge-Based Systems* 25.1 (2012), pp. 13–21 (cit. on p. 37).

[HB02] T. K. Ho, M. Basu. “Complexity measures of supervised classification problems”. In: *IEEE transactions on pattern analysis and machine intelligence* 24.3 (2002), pp. 289–300 (cit. on pp. 14, 34).

[HBV01] M. Halkidi, Y. Batistakis, M. Vazirgiannis. “On clustering validation techniques”. In: *Journal of intelligent information systems* 17.2 (2001), pp. 107–145 (cit. on p. 20).

[HG09] H. He, E. A. Garcia. “Learning from imbalanced data”. In: *IEEE Transactions on knowledge and data engineering* 21.9 (2009), pp. 1263–1284 (cit. on p. 13).

[HRKM18] V. Hirsch, P. Reimann, O. Kirn, B. Mitschang. “Analytical approach to support fault diagnosis and quality control in End-Of-Line testing”. In: *Procedia CIRP* 72 (2018), pp. 1333–1338 (cit. on p. 13).

[HRM20] V. Hirsch, P. Reimann, B. Mitschang. “Exploiting domain knowledge to address multi-class imbalance and a heterogeneous feature space in classification tasks for manufacturing data”. In: *Proceedings of the VLDB Endowment* 13.12 (2020), pp. 3258–3271 (cit. on pp. 13, 14, 25, 26, 29, 30, 34, 39, 55).

[HW79] J. A. Hartigan, M. A. Wong. “Algorithm AS 136: A k-means clustering algorithm”. In: *Journal of the royal statistical society. series c (applied statistics)* 28.1 (1979), pp. 100–108 (cit. on pp. 17, 41).

[HYS+17] G. Haixiang, L. Yijing, J. Shang, G. Mingyun, H. Yuanyue, G. Bing. “Learning from class-imbalanced data: Review of methods and applications”. In: *Expert systems with applications* 73 (2017), pp. 220–239 (cit. on p. 27).

[HYY+16] G. Haixiang, L. Yijing, L. Yanan, L. Xiao, L. Jinling. “BPSO-Adaboost-KNN ensemble learning algorithm for multi-class imbalanced data classification”. In: *Engineering Applications of Artificial Intelligence* 49 (2016), pp. 176–193 (cit. on p. 28).

[Jai10] A. K. Jain. “Data clustering: 50 years beyond K-means”. In: *Pattern recognition letters* 31.8 (2010), pp. 651–666 (cit. on p. 17).

[JD88] A. K. Jain, R. C. Dubes. *Algorithms for clustering data*. Prentice-Hall, Inc., 1988 (cit. on pp. 17, 20).

[KBT11] G. Köksal, I. Batmaz, M. C. Testik. “A review of data mining applications for quality improvement in manufacturing industry”. In: *Expert systems with Applications* 38.10 (2011), pp. 13448–13467 (cit. on p. 13).

[KHS+21] S. K. Karmaker, M. M. Hassan, M. J. Smith, L. Xu, C. Zhai, K. Veeramachaneni. “AutoML to Date and Beyond: Challenges and Opportunities”. In: *ACM Computing Surveys (CSUR)* 54.8 (2021), pp. 1–36 (cit. on pp. 14, 22).

[Kra16] B. Krawczyk. “Learning from imbalanced data: open challenges and future directions”. In: *Progress in Artificial Intelligence* 5.4 (2016), pp. 221–232 (cit. on p. 25).

[LD13] R. Longadge, S. Dongre. “Class imbalance problem in data mining review”. In: *arXiv preprint arXiv:1305.1707* (2013) (cit. on p. 27).

[LJD+17] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, A. Talwalkar. “Hyperband: A novel bandit-based approach to hyperparameter optimization”. In: *The Journal of Machine Learning Research* 18.1 (2017), pp. 6765–6816 (cit. on p. 22).

[MMM13] A. M. Mehar, K. Matawie, A. Maeder. “Determining an optimal value of K in K-means clustering”. In: *2013 IEEE International Conference on Bioinformatics and Biomedicine*. IEEE. 2013, pp. 51–55 (cit. on p. 17).

[Moo96] T. K. Moon. “The expectation-maximization algorithm”. In: *IEEE Signal processing magazine* 13.6 (1996), pp. 47–60 (cit. on p. 19).

[MR17] A. More, D. P. Rana. “Review of random forest classification techniques to resolve data imbalance”. In: *2017 1st International Conference on Intelligent Systems and Information Management (ICISIM)*. IEEE. 2017, pp. 72–78 (cit. on p. 28).

[MRA20] R. Mohammed, J. Rawashdeh, M. Abdullah. “Machine learning with oversampling and undersampling techniques: overview study and experimental results”. In: *2020 11th international conference on information and communication systems (ICICS)*. IEEE. 2020, pp. 243–248 (cit. on pp. 13, 27).

[OIL17] J. Ortigosa-Hernández, I. Inza, J. A. Lozano. “Measuring the class-imbalance extent of multi-class problems”. In: *Pattern Recognition Letters* 98 (2017), pp. 32–38 (cit. on p. 37).

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (cit. on pp. 41, 42).

[Ras03] C. E. Rasmussen. "Gaussian processes in machine learning". In: *Summer school on machine learning*. Springer. 2003, pp. 63–71 (cit. on pp. 19, 41).

[Rok05] L. Rokach. "Ensemble methods for classifiers". In: *Data mining and knowledge discovery handbook*. Springer, 2005, pp. 957–980 (cit. on p. 62).

[Rok10] L. Rokach. *Pattern classification using ensemble methods*. Vol. 75. World Scientific, 2010 (cit. on p. 62).

[Rou87] P. J. Rousseeuw. "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis". In: *Journal of computational and applied mathematics* 20 (1987), pp. 53–65 (cit. on p. 21).

[SB01] S. M. Savaresi, D. L. Boley. "On the performance of bisecting K-means and PDDP". In: *Proceedings of the 2001 SIAM International Conference on Data Mining*. SIAM. 2001, pp. 1–14 (cit. on pp. 19, 41).

[SJ03] C. A. Sugar, G. M. James. "Finding the number of clusters in a dataset: An information-theoretic approach". In: *Journal of the American Statistical Association* 98.463 (2003), pp. 750–763 (cit. on p. 17).

[Ste13] J. Stefanowski. "Overlapping, rare examples and class decomposition in learning classifiers from imbalanced data". In: *Emerging paradigms in machine learning*. Springer, 2013, pp. 277–306 (cit. on p. 25).

[SWK09] Y. Sun, A. K. Wong, M. S. Kamel. "Classification of imbalanced data: A review". In: *International journal of pattern recognition and artificial intelligence* 23.04 (2009), pp. 687–719 (cit. on p. 27).

[TFS] D. Tschechlov, M. Fritz, H. Schwarz. "AutoML4Clust: EfficientAutoML forClustering Analyses". In: () (cit. on pp. 14, 22, 38).

[Tha20] A. Tharwat. "Classification assessment methods". In: *Applied Computing and Informatics* (2020) (cit. on p. 23).

[TTA17] M. Taamneh, S. Taamneh, S. Alkheder. "Clustering-based classification of road traffic accidents using hierarchical clustering and artificial neural networks". In: *International journal of injury control and safety promotion* 24.3 (2017), pp. 388–395 (cit. on p. 29).

[VD20] C.-M. Vong, J. Du. "Accurate and efficient sequential ensemble learning for highly imbalanced multi-class data". In: *Neural Networks* 128 (2020), pp. 268–278 (cit. on p. 28).

[VSC+12] M. Verma, M. Srivastava, N. Chack, A. K. Diswar, N. Gupta. "A comparative study of various clustering algorithms in data mining". In: *International Journal of Engineering Research and Applications (IJERA)* 2.3 (2012), pp. 1379–1384 (cit. on p. 17).

[Wan11] S. Wang. "Ensemble diversity for class imbalance learning". PhD thesis. University of Birmingham, 2011 (cit. on p. 13).

[WCZ+19] J. Wu, X.-Y. Chen, H. Zhang, L.-D. Xiong, H. Lei, S.-H. Deng. “Hyperparameter optimization for machine learning models based on Bayesian optimization”. In: *Journal of Electronic Science and Technology* 17.1 (2019), pp. 26–40 (cit. on p. 22).

[WTMH21] M. Wever, A. Tornede, F. Mohr, E. Hullermeier. “AutoML for multi-label classification: Overview and empirical evaluation”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021) (cit. on pp. 14, 22, 40).

[WY12] S. Wang, X. Yao. “Multiclass imbalance problems: Analysis and potential solutions”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 42.4 (2012), pp. 1119–1130 (cit. on pp. 23, 25, 27).

[ZL10] Z.-H. Zhou, X.-Y. Liu. “On multi-class cost-sensitive learning”. In: *Computational Intelligence* 26.3 (2010), pp. 232–257 (cit. on p. 27).

[ZRL96] T. Zhang, R. Ramakrishnan, M. Livny. “BIRCH: an efficient data clustering method for very large databases”. In: *ACM sigmod record* 25.2 (1996), pp. 103–114 (cit. on pp. 18, 41).

[ZWGS10] L. Zhou, L. Wang, X. Ge, Q. Shi. “A clustering-Based KNN improved algorithm CLKNN for text classification”. In: *2010 2nd International Asia Conference on Informatics in Control, Automation and Robotics (CAR 2010)*. Vol. 3. IEEE. 2010, pp. 212–215 (cit. on p. 29).

All links were last followed on June 30, 2022.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

place, date, signature