Fabio Kazuo Hashimoto de Barros

An Evaluation of Data Resampling Techniques in Local-Model Hierarchical Classification Approaches

Curitiba - PR, Brasil

2024

Fabio Kazuo Hashimoto de Barros

An Evaluation of Data Resampling Techniques in Local-Model Hierarchical Classification Approaches

Dissertation presented to the Graduate Program in Informatics (PPGIa) of the Pontifícia Universidade Católica do Paraná (PUCPR) as partial fulfillment of the requirements for the degree of Master in Computer Science.

Pontifícia Universidade Católica do Paraná (PUCPR)

Graduate Program in Informatics (PPGIa)

Supervisor: Prof. Dr. Carlos Nascimento Silla Jr.

Curitiba - PR, Brasil

2024

Abstract

In real world data, there are some problems where the class structure has an inherent hierarchical relationship between the classes. In these scenarios, it makes sense to model the problem using a hierarchical classification approach. The task of hierarchical classification is defined as a task where the class structure of a dataset can be organized in a taxonomy or hierarchy. A problem that is often observed in real world data is the imbalanced class distribution, and this problem also appears in hierarchical classification. The imbalanced class distribution occurs when there are classes that have a lot more samples than others. This phenomenon causes the class distribution to be skewed towards the classes that occur more often, the majority classes. This unequal distribution distribution negatively affects the classification performance because the classifier tends to benefit the majority class. Because of this problem, our objective is to evaluate the effectiveness of using different data resampling approaches in imbalanced hierarchical datasets from different application domains. To do this, we test well known resampling approaches, such as the flat resampling, and we also propose new methods, using them with different kinds of local hierarchical classifiers and resampling algorithms from the literature. Our choice to use local hierarchical classifiers is because there are few works in the literature that investigate imbalanced distributions in local hierarchical classification problems. After testing both the well known data resampling approoaches and the proposed ones, we evaluate their effectiveness compared to a baseline, where no resampling is used. Our results reported that data resampling yields statistically significant improvement to the classification performance and also that one of the proposed data resampling approaches was the best ranked approach in one of the local hierarchical classification scenarios.

Keywords: Hierarchical classification, Imbalanced Learning, Class Imbalance, Resampling algorithms.

List of Figures

List of Tables

List of Algorithms

List of abbreviations and acronyms

ADASYN Adaptive Synthetic Sampling

CD Critical Difference

CNN Convolutional Neural Networks

CT Computed Tomography

CXR Chest X-Ray

CSV Comma Separated Values

DAG Directed Acyclic Graph

DT Decision Trees

ENN Edited Nearest Neighbor

FD Full Depth

HMC Hierarchical Multi-Label Classification

IDC Image Data Collection

IR Imbalance Ratio

kNN K-Nearest Neighbors

LBP Local Binary Patterns

LCL Local Classifier per Level

LCN Local Classifier per Node

LCPN Local Classifier per Parent Node

MLNP Mandatory Leaf Node Prediction

MLP Multilayer Perceptron

MPL Multiple Path of Labels

MPP Multiple Path Prediction

NB Naive Bayes

NMLNP Non-Mandatory Leaf Node Prediction

PD Partial Depth

RF Random Forest

ROS Random Oversampling

SMOTE Synthetic Minority Oversampling Technique

SPL Single Path of Labels

SPP Single Path Prediction

SVM Support Vector Machine

WIPO World Intellectual Property Organization

List of symbols

$Tree$: Represents the tree of local classifiers

Lc_i: They represent the i^{th} local classifier object, composed by $data$ (the subset of data of the given local classifier) and $lblPath$ (the label path for the given local classifier)

D: Represents the original dataset

CV: Represents the the list of Cross Validation splits

Tr: Training folds of data

Ts: Test fold of data

R: List of resampling algorithms

clf: Best classifier obtained from the resampler selection procedure

$best$: Variable that stores the best macro-averaged F1 score

$bestClf$: Variable that stores the classifier with the best score

STr: Sub-train set, obtained after splitting the local classifier data in sub-train and validation

STr': Sub-train set after resampling

Vs: Validation set, obtained after splitting the local classifier data in sub-train and validation

Contents

1 Introduction

The interest on Machine Learning applications in real world problems has been experiencing a big increase in the past decade. The expectations on this field of study have never been higher, and it is expected that the Machine Learning field will continue to experience a substantial growth in the next few years (COLUMBUS, 2020).

One of the main areas of study in Machine Learning is the supervised learning task, also known as, classification. Even with the classification algorithms, there are different types of problems that they can handle, such as:

- Binary: In this type of classification, the task is to assign a given data example to one out of two classes (PARMIGIANI, 2001).
- Multi-class: In this type of approach, the task is to assign each data example to one label from a set of previously defined labels (SAHARE; GUPTA, 2012).
- Multi-label: In this approach, each data example is associated with a set of labels from a set of previously defined labels (TSOUMAKAS; KATAKIS, 2007).
- Hierarchical Classification: The main difference between hierarchical classification algorithms and the previous ones, is that, according to (SILLA; FREITAS, 2011), the outputs of these classifcation problems are defined according to a class taxonomy or hierachy. These classes are connected to each other through the IS-A relationship.

Nowadays, classification models may apply to a diverse range of domains, and when we deal with real world problems, there is a challenge for the Data Mining community that many researchers face: the class imbalance problem (FERNÁNDEZ et al., 2013). This happens mainly because real world datasets usually have classes that occur more frequently (majority classes) and other ones that have fewer occurrences (known as minority classes). The classification algorithms usually benefit the majority classes at the expense of the minority ones. However, sometimes our main interest is to label the rare patterns instead of the most frequent ones, such as in credit card fraud detection (KUMAR et al., 2015) and medical image classification (PEREIRA et al., 2020; BAI et al., 2019; ARIAS et al., 2016; ABDULRAZZAQ et al., 2020).

The main issue with imbalanced datasets used in standard classification algorithms is that they are suitable for balanced scenarios, but when there is a class imbalance, it is very likely that the classifier will be biased towards the majority classes (FERNÁNDEZ et al., 2013). Because of this, the classifier will have a limited capacity to identify the

samples from the minority classes, which could be problematic in some domains where the main interest is to identify the classes that represent rare patterns.

A good example where this kind of issue may arise is in the medical imaging domain. Imagine a scenario where the majority class has examples labeled as "healthy" and the minority class has examples labeled as "not healthy". If we train the classifier with this dataset, without any sort of approach to handle the class imbalanceness, it would be biased towards the "healthy" examples, so there is a higher probability that the classifier would predict as a "healthy" sample. The cost of misclassification in this scenario is very high, once that if we diagnose a "not healthy" patient as "healthy", that could have very serious consequences.

According to (LOYOLA-GONZÁLEZ et al., 2016), there are also multiple approaches used in the literature to deal with the class imbalanceness issue. The most well known are: data-level, algorithmic-level and cost-sensitive. The data-level approach is considered the most versatile approach, because it is applied only to the dataset which makes it independent of the classification algorithm being used. The most widely used data-level technique is called data resampling, which consists in creating or removing data examples from the dataset in order to make the class distribution balanced.

The imbalanced data issue is something that could also happen in hierarchical classification scenarios. Because of this, we propose to conduct a study to assess the effectiveness of data resampling applied to the hierarchical classification task, in multiple domains. In this work, we aim to test well known data resampling approaches and also to propose new ones. Our main interest is to establish a baseline where no resampling is used and then compare it with the results obtained using different data resampling approaches to evaluate how effective these techniques are.

1.1 Objectives

To cope with the imbalanced class distribution problem that negatively affects the hierarchical classification performance, the main objective of this work is to evaluate the use of different data resampling approaches for the hierarchical classification task, using local classifiers, and compare their effectiveness with a baseline. To achieve this goal, we define following specific objectives:

1. Establish a baseline where no data resampling approach is used.

2. Evaluate the use of resampling algorithms in hierarchical classification datasets (flat resampling).

3. Evaluate the use of resampling algorithms in hierarchical classification datasets,

considering the local class distributions at each local classifier and applying the resampling algorithms at local level (local resampling).

4. Evaluate the use of resampling algorithms in hierarchical classification datasets, considering the local class distributions at each local classifier and applying the resampling algorithms at local level for selected classes (threshold selective resampling).

5. Evaluate the use of resampling algorithms in hierarchical classification datasets, considering the local class distributions at each local classifier and applying the different resampling algorithms for each one of local classifiers (local selective resampling).

1.2 Hypothesis

- ***Hypothesis***: The proposed resampling approaches outperform the baseline approach (without resampling) for hierarchical classification problems.

1.3 Contributions

This work aims to contribute to the scientific community with an evaluation and analysis of how the performance of local hierarchical classifiers is affected when data resampling techniques are applied to hierarchical datasets. It is noteworthy that some of the results obtained during the development of this work were used in a paper that was accepted and presented at the 21st IEEE International Conference on BioInformatics and BioEngineering (BARROS et al., 2021).

We also proposed new resampling approaches, the Threshold Selective and the Local Selective approaches, with the objective to improve the hierarchical classification results obtained with existent approaches, such as the Flat Resampling approach.

Besides contributing scientifically and technically, we used this effort to bring positive impact to the society as well, by participating of a competition sponsored by the government of the state of São Paulo, where the objective was to develop a solution using machine learning to help in the COVID-19 identification through Chest X-Ray (CXR) images (Secretaria de Desenvolvimento Econômico, 2020).

1.4 Document Structure

The structure of this document is organized as it follows: chapter 2 provides us theoretical background on the hierarchical classification and imbalanced data topics. In chapter 3, we provide a review and summary on the recent works in the literature that

treated the class imbalanceness issue for hierarchical classification tasks in a similar way. The next section talks about what is the methodology we developed to implement our proposals. In chapter 5, we present and analyze the results. Finally, in chapter 6, the concluding remarks are discussed, considering the results that were obtained.

2 Theoretical Background

In this chapter, the main concepts used throughtout this work are discussed. In section 2.1, the conceptual knowledge of the hierarchical classification task is presented and explained. In this work we will be focusing in the local classifiers and because of that we will not discuss the global classifiers in detail. Section 2.4 presents the most used approaches to resolve the class imbalance issue and some of the most commonly used resampling algorithms.

2.1 Hierarchical Classification

The hierarchical classification task, a type of supervised learning, could be briefly defined as a classification task where its classes are hierarchically related to each other, making it possible to represent their relationship using a taxonomic structure. In this task, we are considering exclusively those problems where this hierarchy is pre-defined, which means that it is inherent to the problem.

In order to have a clearer understanding about how these structures are organized, we further detail the definitions and terms from the hierarchical classification domain in the section 2.2.

2.2 Definitions and Terminology

The first important definition that we need to understand the hierarchical classification task is the definition of taxonomy. Let us consider a tree structured hierarchy where the set of all the possible classes in this problem is represented by C and the $\prec$ symbol represents the "IS-A Relationship". According to (SILLA; FREITAS, 2011), this relationship may be considered as asymmetric, anti-reflexive and transitive:

- The greatest element "R" is the root of the tree.
- $\forall c_i, c_j \in C$, if $c_i \prec c_j$ then $c_j \not\prec c_i$
- $\forall c_i \in C, c_i \not\prec c_i$
- $\forall c_i, c_j, c_k \in C, c_i \prec c_j$ and $c_j \prec c_k$ imply $c_i \prec c_k$

If the class structure of the problem being studied satisfies the aforementioned properties, that means that it is a hierarchical classification problem. These four properties

were tailored for tree structures, but it is also adequate for problems that are modeled with a Directed Acyclic Graph (DAG).

After evaluating if the class structure represents a hierarchical classification problem and confirming that it does, based on (SILLA; FREITAS, 2011), we can also categorize this problem according to the following features:

- The type of structure representing the hierarchical structure
  - Tree: which means the classes are arranged in a tree structure.
  - Graph: which means the classes are arranged in a DAG structure.
- If a data instance is allowed to be associated with a single or multiple paths of the hierarchy
  - Single Path of Labels (SPL): means that the instance may have only one path of labels down the hierarchy.
  - Multiple Path of Labels (MPL): means that the instance may have more than one path of labels down the hierarchy. It is a synonym of the term "hierarchically multi-label".
- The label depth for each data instance
  - Partial Depth (PD): indicates that the labels for each data instance may be intermediate classes, without reaching the leaf nodes.
  - Full Depth (FD): indicates that all the labels for all data instances are leaf nodes.

According to (SILLA; FREITAS, 2011), we can also categorize the classification algorithm that will be produced to handle this hierarchical classification problem. Note that, even though some of them sound familiar to the previous categorization, they refer to different entities (algorithms vs problems). They are the following ones:

- Whether the algorithm will be capable to predict label in a single or multiple paths in the hierarchy.
  - Single Path Prediction (SPP): The algorithm is capable of associating each data instance with a single path of labels.
  - Multiple Path Prediction (MPP): The algorithm is capable of associating each data instance with multiple paths of labels.
- The prediction depth of the algorithm

  - Mandatory Leaf Node Prediction (MLNP): Indicates that the algorithm will always predict leaf classes.
  - Non-Mandatory Leaf Node Prediction (NMLNP): Indicates that the algorithm is able to predict classes at any level, including leaf ones.

- Which taxonomy the algorithm is able to use
  - Tree: means that the algorithm can handle classes that are arranged in a tree.
  - DAG: means that the algorithm can handle classes that are arranged in a DAG.

- The algorithm type
  - LCN: For this particular case, it is important to specify which strategy will be used to select the positive and negative examples. These strategies are also called policies, as mentioned below:
    * Exclusive
    * Less Exclusive
    * Less Inclusive
    * Inclusive
    * Siblings
    * Exclusive Siblings
  - LCPN
  - LCL
  - Global Classifier

In order to categorize a real world problem and the algorithm to be used, let us consider a sample dataset based on a real-life dataset. In this sample dataset, we use as an example a pneumonia detection dataset with chest x-ray (CXR) images that represent healthy or unhealthy lungs. The healthy images come from the RYDLS dataset (PEREIRA et al., 2020), and the unhealthy ones come from the C-19 IDC (COHEN et al., 2020) dataset. This dataset has data instances with SPL and FD. The final hierarchical structure is the tree presented in figure 1, and the final distribution that we will use throughout our examples is presented in table 1.

2.3 Local Classifier Approaches

The local classification approaches are among the most common hierarchical classification approaches in the literature. The main idea behind them is to take the underlying hierarchy of the problem into account, to use the information with a local

Table 1 – Class labels for a Sample dataset based on the RYDLS C-19 IDC dataset

Sample Dataset	
Class	**Count**
R/Normal	1000
R/Viral/COVID-19	479
R/Fungal/Pneumocystis	30
R/Bacterial/Streptococcus	22
R/Viral/SARS	16
R/Lipoid	13
R/Bacterial/Mycoplasma	11
R/Bacterial/Klebsiella	10
R/Bacterial/Legionella	8

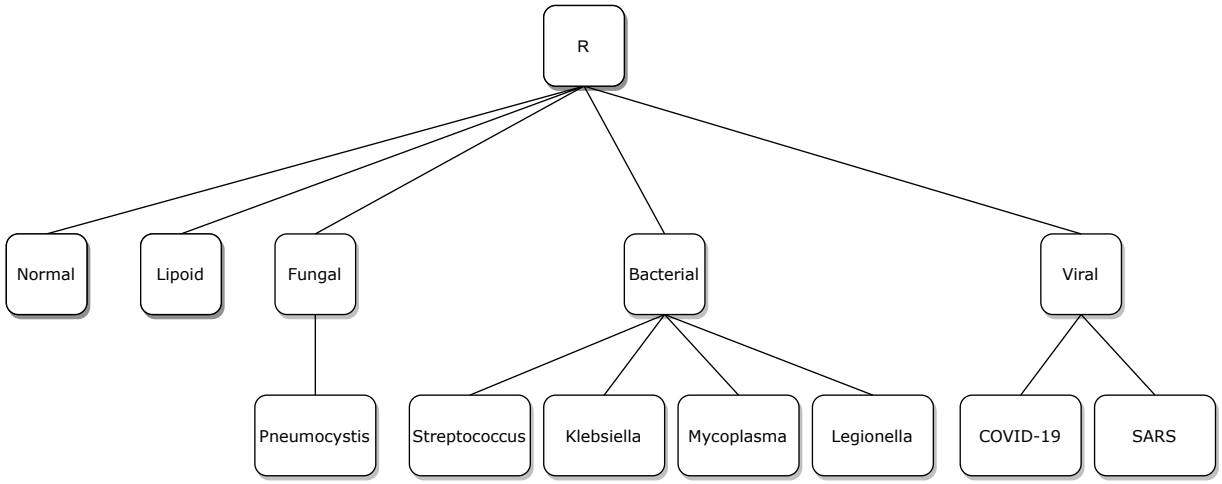


Figure 1 – Taxonomy produced from the class labels in table 1

perspective. There are three well-known classifier types that use this local information from three different perspectives: the local classifier per node (LCN), the local classifier per parent node (LCPN) and the local classifier per level (LCL). These three approaches are explained in more detail in the next sections.

2.3.1 Local Classifier per Node (LCN)

This is one of the most widely used local approaches in the hierarchical classification domain. Basically, it is consisted of one binary classifier for each class in the hierarchy. The figure 2 presents an example of how this approach would work for the hierarchy presented in figure 1. As we can see, each of the nodes will contain a binary classifier which will predict the samples as belonging to the positive examples or the negative ones.

Also, as mentioned in section 2.2, the distinction between the positive and negative examples for each of these classifiers is done using policies. Let us use the terminology in table 2 defined by (SILLA; FREITAS, 2011) to define each policy.

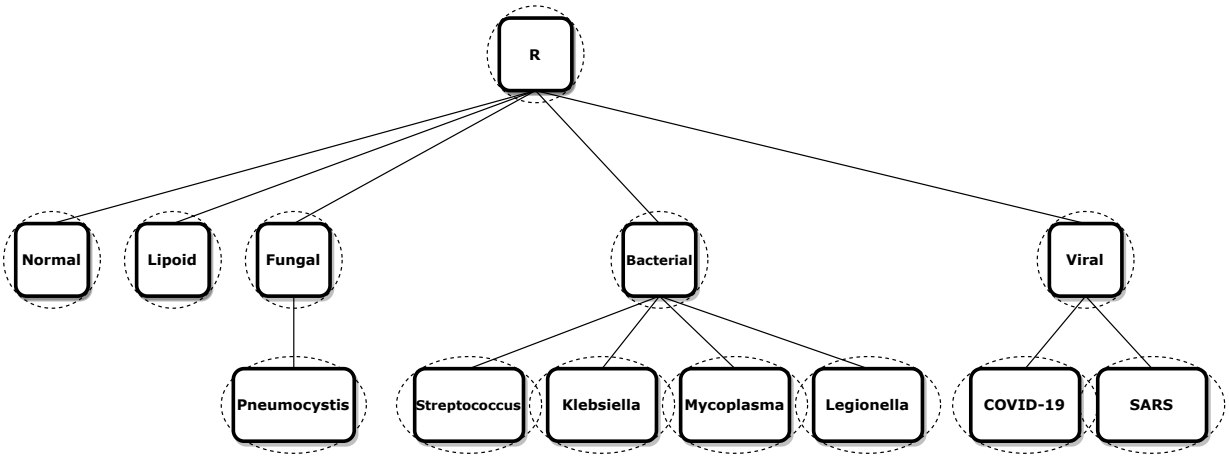


Figure 2 – LCN example based on the hierarchy produced in figure 1

Table 2 – Notation used in the definition of each policy

Symbol	**Meaning**
T_r	All training examples
$T_r^+(c_j)$	Positive training examples of c_j
$T_r^-(c_j)$	Negative training examples of c_j
$\uparrow(c_j)$	The parent of c_j
$\downarrow(c_j)$	The set of children of c_j
$\Uparrow(c_j)$	The set of ancestors of c_j
$\Downarrow(c_j)$	The set of descendants of c_j
$\leftrightarrow(c_j)$	The set of siblings of c_j
$*(c_j)$	Examples whose the most specific class is c_j

- Exclusive: In this policy, $T_r^+(c_j)$ is represented only by $*(c_j)$, while $T_r^-(c_j) = \setminus *(c_j)$. In this example, let us consider the hierarchy in figure 1. If we assume c_j = Viral class, we would have $T_r^+(c_j)$ = Viral and $T_r^-(c_j)$ = Viral/COVID-19, Viral/SARS, Bacterial, Bacterial/Streptococcus, Bacterial/Klebsiella, Bacterial/Mycoplasma, Bacterial/Legionella, Normal, Lipoid and Fungal/Pneumocystis. This may sound inconsistent because the set of children of c_j is marked as negative.

- Less Exclusive: The inconsistency observed in the previous policy is resolved here. The positive examples $T_r^+(c_j) = *(c_j)$ while the negative examples $T_r^-(c_j) = \setminus *(c_j) \cup \Downarrow(c_j)$. In this case, we would have $T_r^+(c_j)$ = Viral and $T_r^-(c_j)$ = Bacterial, Bacterial/Streptococcus, Bacterial/Klebsiella, Bacterial/Mycoplasma, Bacterial/Legionella, Normal, Lipoid and Fungal/Pneumocystis.

- Less Inclusive: In this policy, the descendant examples of class c_j are included as part of the positive examples. $T_r^+(c_j) = *(c_j) \cup \Downarrow(c_j)$ and $T_r^-(c_j) = \setminus *(c_j) \cup \Downarrow(c_j)$. In our example, the positive examples would be $T_r^+(c_j)$ = Viral, Viral/COVID-19, Viral/SARS and $T_r^-(c_j)$ = Bacterial, Bacterial/Streptococcus, Bacterial/Klebsiella,

Bacterial/Mycoplasma, Bacterial/Legionella, Normal, Lipoid and Fungal/Pneumocystis.

- Inclusive: The only difference compared to the "Less Inclusive" policy is that, for this one, any ancestors $\Uparrow(c_j)$ are excluded from the negative examples. Thus, we have $T_r^+(c_j) = *(c_j) \cup \Downarrow(c_j)$ and $T_r^-(c_j) = \setminus * (c_j) \cup \Downarrow(c_j) \cup \Uparrow(c_j)$. In our example, we would need to remove any ancestors of the Viral class from the negative examples, if there were any.

- Siblings: In the siblings policy, the positive examples are consisted of the c_j class and its descendants while the negative ones are consisted of the c_j siblings and its descendants. Thus, we have: $T_r^+(c_j) = *(c_j) \cup \Downarrow(c_j)$ and $T_r^-(c_j) = \leftrightarrow *(c_j) \cup \Downarrow(\leftrightarrow(c_j))$. In our example, $T_r^+(c_j)$ = Viral, Viral/COVID-19, Viral/SARS and $T_r^-(c_j)$ = Bacterial, Bacterial/Streptococcus, Bacterial/Klebsiella, Bacterial/Mycoplasma, Bacterial/Legionella, Normal, Lipoid and Fungal/Pneumocystis.

- Exclusive Siblings: The main difference between this policy and the Siblings one is that we do not take into account the descendants of c_j. Thus, we would have: $T_r^+(c_j) = *(c_j)$ and $T_r^-(c_j) = \leftrightarrow *(c_j)$. In our example, the positive examples would be from the Viral class and the negative ones would belong to the Bacterial, Normal, Lipoid and Fungal classes.

2.3.1.1 Training Phase

The training phase of a LCN classifier is done independently of the hierarchical structure being used, we traverse the Tree or DAG passing through all the nodes and training the binary classifiers in each one of them. The training process ends when we reach the end of the structure.

2.3.1.2 Prediction

Differently from what was discussed in the training phase, the prediction process in the LCN is far more complicated. A common approach used in the hierarchical classification domain is the top-down approach.

This approach was originally proposed in (KOLLER; SAHAMI, 1997). Basically, the prediction at each level l of the hierarchy (except at root level) entirely depends on the prediction made at the previous (or parent) level $l-1$. Using the hierarchy produced in figure 1 as an example, let's say that at level 1, the output for the Bacterial classifier is true and all the other ones are false. The next level will narrow down the choices only to those classes that are a child of the Bacterial node. This means that only the Streptococcus, Klebsiella, Mycoplasma and Legionella classes will be considered in the next prediction at

level 2. An example predicting a data instance from class Streptococcus is shown in figure 3.

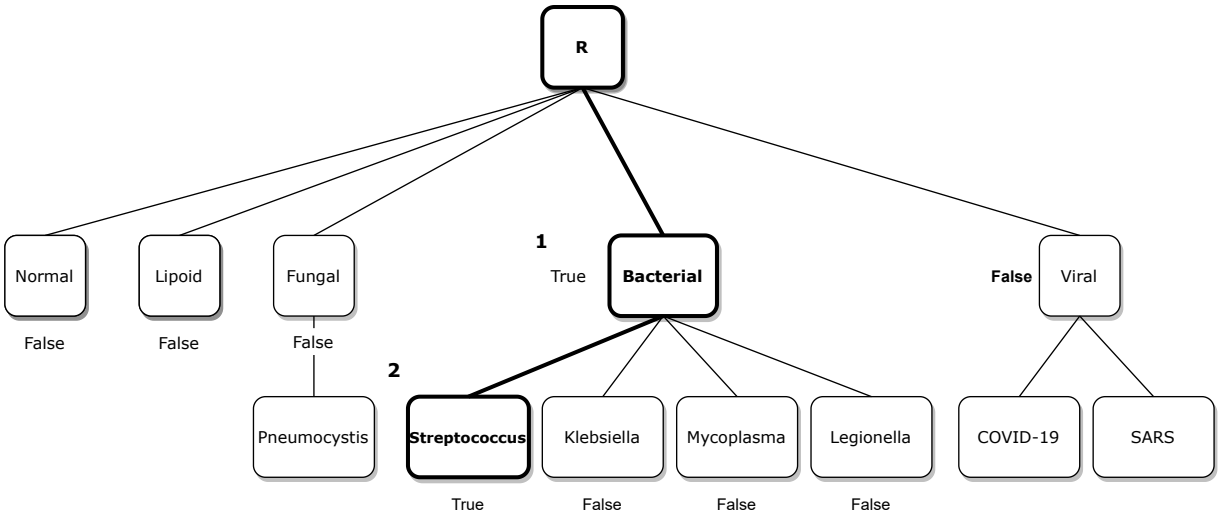


Figure 3 – Prediction using the top-down approach for the LCN scenario

Even tough the top-down approach solves the inconsistency problems of classifying a data instance as Viral/Streptococcus in our example, there is also another problem when this approach is used together with the LCN classifier. What if we are dealing with a NMLNP problem? In this case, we need a criterion to stop the prediction before it reaches the leaf levels. Another issue that may arise at this point is if all the classifiers at a certain level return us a false prediction. If that happens, we would face a "blocking" problem. In figure 3, imagine if all the binary classifiers from level 1 had a false response. We would not be able to continue the prediction to level 2 if the problem is MLNP, or, if we are dealing with a NMLNP problem, we would not have a predicted class. This issue will be further elaborated in section 2.3.4.

2.3.2 Local Classifier per Parent Node (LCPN)

Another approach found in the literature that copes with local information in the hierarchical classification dominain is the LCPN. The main difference between the LCPN and the LCN is that the LCPN uses a multi-class classifier for each parent node in the class hierarchy. These classifiers are trained to distinguish data instances that belong to classes of any of its child nodes (SILLA; FREITAS, 2011). Figure 4 presents an example of the hierarchy for this scenario. Each of the nodes that were highlighted will have a multi-class classifier. An important observation here is that, from an implementation perspective, the Fungal class will not have a classifier because it has a single child class.

In the same way as the LCN, this approach also needs some policies to determine the positive and negative classes for each multi-class classifier. The "siblings" and "exclusive-

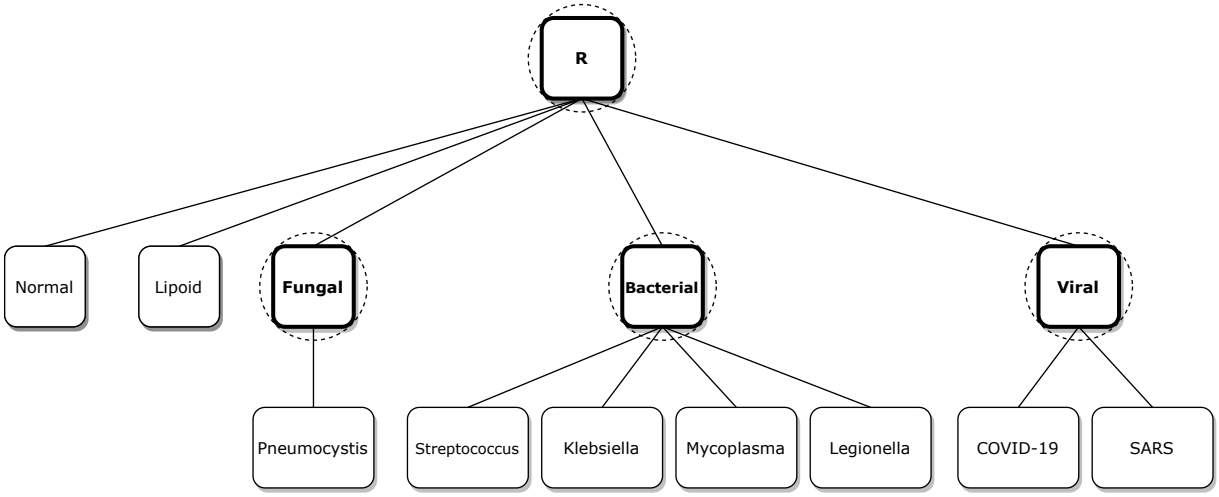


Figure 4 – LCPN example based on the hierarchy produced in figure 1

siblings" policies, mentioned in section 2.3.1, are both adequate to be used with the LCPN approach.

2.3.2.1 Training Phase

The training process of the LCPN is very similar to the one described for the LCN. The hierarchical structure is traversed, passing through all the nodes. For each parent node, we will have a multi-class classifier which will be trained with the positive examples for that node. The training algorithm ends when the last parent node is reached.

2.3.2.2 Predict

The prediction process for the LCPN using the top-down approach shares similarities with the LCN process. The main difference here is that we will have multi-class classifiers instead of binary ones. Figure 5 shows an example of the top-down prediction considering the LCPN scenario. In this case, a data instance from class Bacterial/Streptococcus is being predicted.

In this case, the prediction starts at root level. The root classifier predicted a Bacterial class, then the process continues in the Bacterial parent node. This last classifier then predicts that the data instance belongs to the Streptococcus class. Since the problem presented is MLNP and FD, the prediction is then completed.

2.3.3 Local Classifier per Level (LCL)

The LCL consists of using one multi-class classifier for each level in the hierarchy. Figure 6 adapts the example of the dataset presented in table 1 to show a LCL classifier using that hierarchy. As we have two levels, we would have two multi-class classifiers.

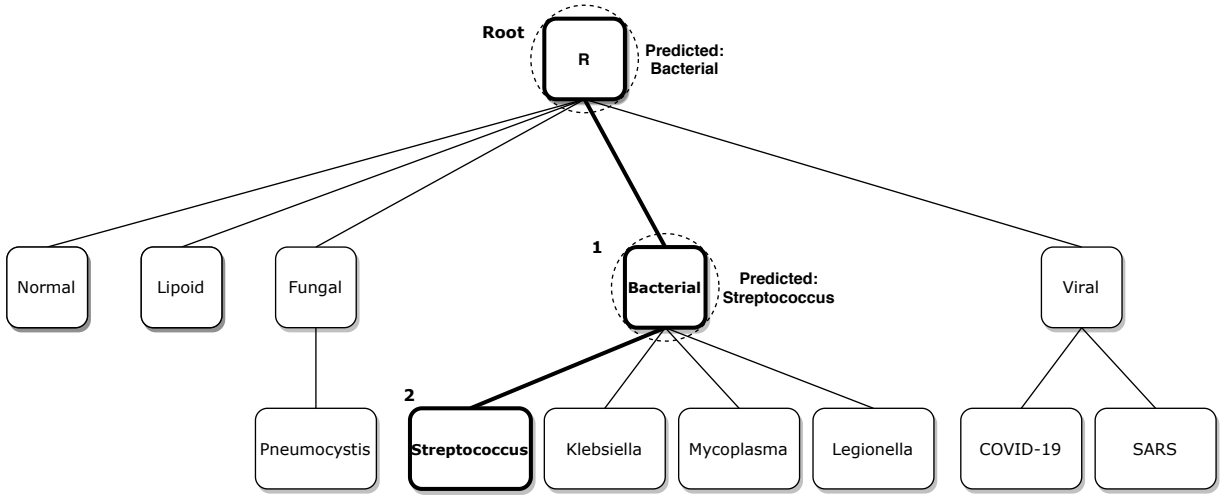


Figure 5 – Top-down approach for the LCPN scenario.

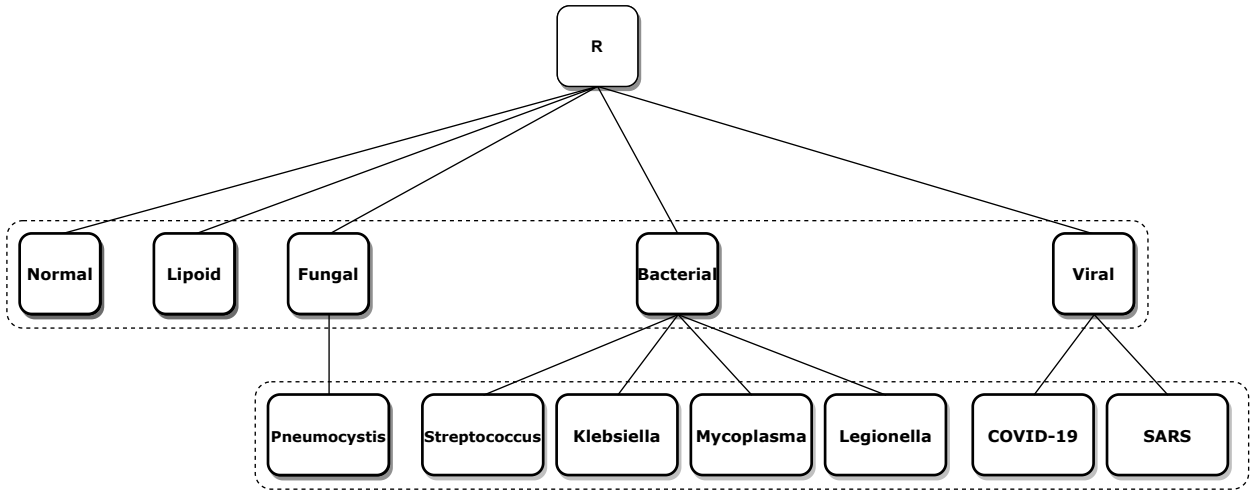


Figure 6 – LCL example based on the hierarchy produced in figure 1

2.3.3.1 Prediction

The prediction for the LCL scenario may also be done using the top-down approach mentioned for both LCPN and LCN scenarios. Similarly to the LCPN classifier, after a prediction is done at a certain level l, only the child classes of the node predicted at level l will be considered for the prediction at level $l + 1$.

Figure 7 shows an example of a top-down prediction using the LCL classifier. In this case, we can see that at the first level of the hierarchy, the predicted class was Bacterial. Then, when we move to the second level. According to the top-down method, the only options available here, are the child nodes of Bacterial: Streptococcus, Klebsiella, Mycoplasma and Legionella. The predicted class was Streptococcus.

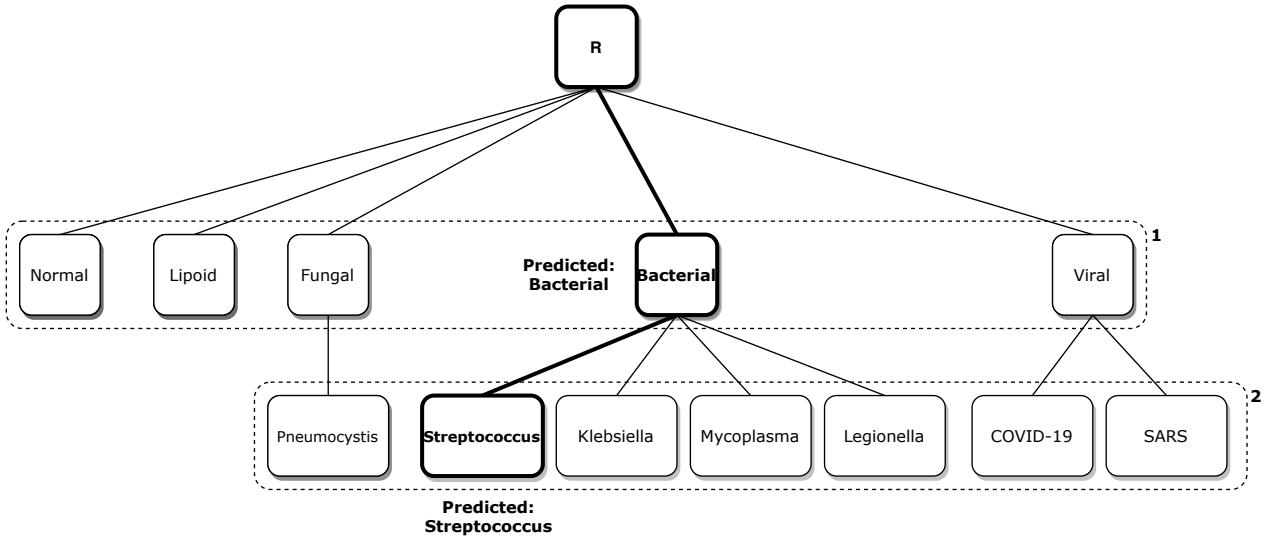


Figure 7 – Top-down prediction for the LCL classifier

2.3.4 The blocking problem

The blocking problem happens during the prediction phase in the top-down approach and it may happen in any of the local classifiers (LCN, LCPN or LCL). Blocking happens when, during the prediction, the classifier predicts that a certain data instance does not have any class associated with it. Because of this, the top-down prediction cannot continue, or, if we have a NMLNP problem, we will not have a label for that data instance. Let us consider the example of the blocking problem applied to the LCN scenario, presented in figure 8.

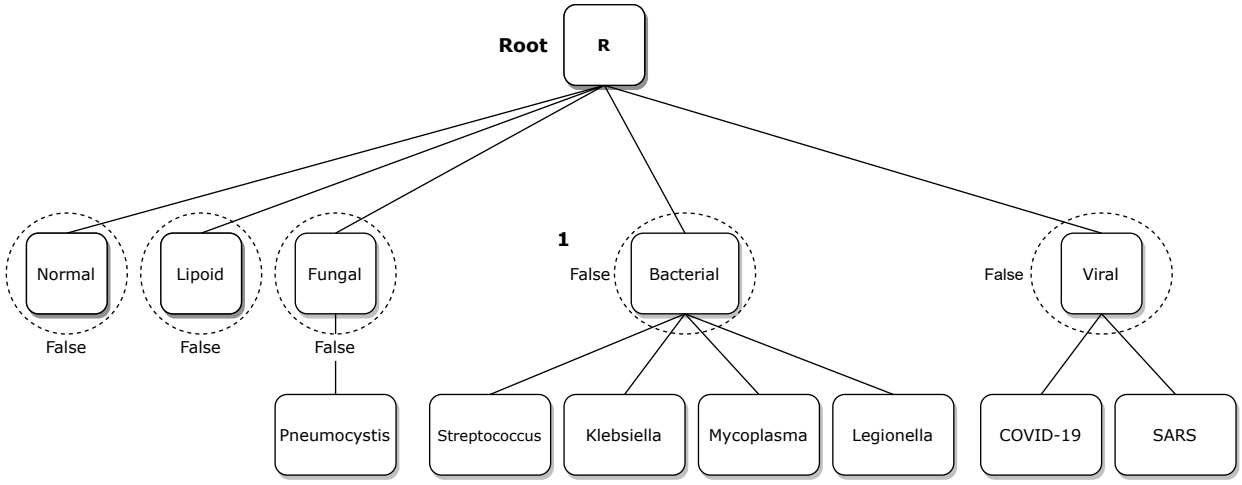


Figure 8 – Blocking problem in the LCN classifier

As we can see, in this case, the predictions at the first level show that the data instance has no class associated with it, because all the binary classifiers did not identify

the sample as belonging to any of the classes at that level. Because of this, the prediction is "blocked" and neither will stop at this level nor continue to the next one. In order to avoid this, there are some strategies to solve this issue, as defined in (SUN et al., 2004):

- Threshold reduction method: Reduce the probability threshold of the classifiers, in order to be able to allow more examples to be passed to the classifiers at lower levels.
- Restricted voting: This method depends on a set of secondary classifiers to link a node to its grandparent. Basically, this set of secondary classifiers will be trained with a different training set, and will act as a second opinion about which class the sample belongs to. To classify a sample with class C, both main and secondary classifiers must agree with each other at grandparent and parent levels.
- Extended multiplicative thresholds: This rule is an extension of the threshold rule defined by (DUMAIS; CHEN, 2000), where the authors propose a rule for a two-level hierarchy. To label an example with a specific class, the posterior probabilities at the first and second levels must be higher than a threshold. In the extended multiplicative version of this rule, we consider the product of the posterior probabilities at the first and second levels, and then compare this product with the threshold.

2.3.5 Evaluation Metrics

According to (SILLA; FREITAS, 2011), there are many works in the literature that use different metrics to evaluate hierarchical classification algorithms. Some of them use flat classification metrics such as precision, recall and f-measure while others even propose their own metrics. However, there is a study, proposed by (KIRITCHENKO et al., 2006) where the author proposes hierarchical classification metrics that shows that using this metrics had a difference of 29.39% in the worst case scenario, compared to the flat evaluation metrics. The main question that comes up is: is it adequate to evaluate a hierarchical classification model using metrics that are suited for flat classification models? What if we evaluated them using hierarchical measures as well?

If we want to specifically use hierarchical measures to evaluate these models, which ones should we use? Some new adapted metrics were proposed by (KIRITCHENKO et al., 2005). These measures are extensions of the widely used precision, recall and f-measure metrics. Let us define the variables involved in these calculations:

- i: a data instance from the testing data
- $\hat{P}_i$: the set of the most specific classes predicted for instance i
- $\hat{T}_i$: the set of the true most specific classes for instance i

- hP: hierarchical precision
- hR: hierarchical recall
- hF: hierarchical f-measure

Using these definitions, we have:

$$hP = \frac{\sum_i |\hat{P}_i \cap \hat{T}_i|}{|\hat{P}_i|} \tag{2.1}$$

$$hR = \frac{\sum_i |\hat{P}_i \cap \hat{T}_i|}{|\hat{T}_i|} \tag{2.2}$$

$$hF = \frac{2 * hP * hR}{hP + hR} \tag{2.3}$$

To have a better understanding of how these metrics work, let us consider the following examples:

- a) **Predicted Class:** R/Bacterial - **True Known Class:** R/Bacterial/Streptococcus
- b) **Predicted Class:** R/Viral/COVID - **True Known Class:** R/Viral/COVID
- c) **Predicted Class:** R/Viral/SARS - **True Known Class:** R/Viral/COVID

For all these cases, we will ignore the root R. Solving example a), we would have:

$$hP = \frac{\sum |Bacterial \cap Bacterial/Streptococcus|}{\sum |Bacterial|} = \frac{1}{1} = 1 \tag{2.4}$$

$$hR = \frac{\sum |Bacterial \cap Bacterial/Streptococcus|}{\sum |Bacterial/Streptococcus|} = \frac{1}{2} = 0.5 \tag{2.5}$$

$$hF = \frac{2 * 1 * 0.5}{1 + 0.5} = 1/1.5 = 0.67 \tag{2.6}$$

By solving example b), we would have:

$$hP = \frac{\sum |Viral/COVID \cap Viral/COVID|}{\sum |Viral/COVID|} = \frac{2}{2} = 1 \tag{2.7}$$

$$hR = \frac{\sum |Viral/COVID \cap Viral/COVID|}{\sum |Viral/COVID|} = \frac{2}{2} = 1 \tag{2.8}$$

$$hF = \frac{2 * 1 * 1}{1 + 1} = 2/2 = 1 \tag{2.9}$$

And finally, for example c), we have:

$$hP = \frac{\sum |Viral/SARS \cap Viral/COVID|}{\sum |Viral/SARS|} = \frac{1}{2} = 0.5 \tag{2.10}$$

$$hR = \frac{\sum |Viral/SARS \cap Viral/COVID|}{\sum |Viral/COVID|} = \frac{1}{2} = 0.5 \tag{2.11}$$

$$hF = \frac{2 * 0.5 * 0.5}{0.5 + 0.5} = 0.5/1 = 0.5 \tag{2.12}$$

2.4 Imbalanced Data

The imbalanced data issue appears when the number of data instances that belong to the majority class is very different from the amount belonging to the minority one. The simplest way to identify if the dataset is imbalanced is by computing the Imbalance Ratio (IR), defined in (ORRIOLS-PUIG; BERNADÓ-MANSILLA, 2009):

$$IR = \frac{N_{majority}}{N_{minority}} \tag{2.13}$$

If the result of this operation is greater than 1, that means that we have more samples from the majority class than the minority one. The bigger the value becomes, the more imbalanced the dataset is. This kind of issue is common when we use real world data to build classification models once that real world data tends to be imbalanced.

In many fields of study, there is usually a lot more samples that belong to a majority class and a lot less samples belonging to a minority class. Some of the domains where this is common are: medical imaging, anomaly detection, credit card fraud detection and many others. This happens because there is a pattern in these datasets that occurs more often than the others. The minority classes, which have fewer occurrences, are usually associated with the classes of main interest or the most important concepts to be learned. In the medical imaging domain, they are usually associated with pathologies, in credit card detection they are usually associated with fraudulent transactions and in anomaly detection they are associated with anormal behavior.

This inherent imbalance in the class distribution brings us a big issue to the construction of classification models for these datasets. Because of this big difference in the number of examples belonging to the majority class compared to the minority one, the predictions of this trained model will be most likely be biased towards the majority

class, which means that there is a higher chance that the classifier will label the sample as belonging to the majority class.

Because of this, there are multiple approaches to deal with class imbalance. They will be further detailed in section 2.4.1.

2.4.1 Dealing with data imbalanceness

There are multiple approaches to deal with the class imbalance in classification problems. According to (FERNÁNDEZ et al., 2013), these approaches are grouped in three main groups:

- Data level: This approach consists in resampling the data in order to overcome the class imbalance issue. They are mostly based on algorithms that create new data instances based on the existing ones. It may be considered an external approach because it mostly deals with the data distribution of the dataset.
- Algorithmic level approaches: This is a solution that is focused on adapting the classification algorithm to reinforce the learning towards the minority classes. This may be considered as an internal approach, once that it tries to modify the algorithm to make it more suitable for the minority class.
- Cost-sensitive: These solutions are suitable to be applied to data and algorithmic level, or even both used together. The focus of this approach is to minimize the misclassification cost.

The data level approach is considered more versatile because it does not have any dependencies of the classification algorithm, once that we can simply resample the data as a pre-processing step and use it in any classification algorithm. These resampling methods are divided in three sub-groups: undersampling, oversampling and hybrid. Each one of them is defined as it follows, and presented in figure 9.

- Undersampling: consists in eliminating data instances from the majority classes.
- Oversampling: consists in creating synthetic data examples for the minority classes.
- Hybrid: This method is a combination of undersampling and oversampling techniques. It consists in using oversampling to create synthetic examples for the minority classes while also doing some data cleaning in the examples of the majority class.

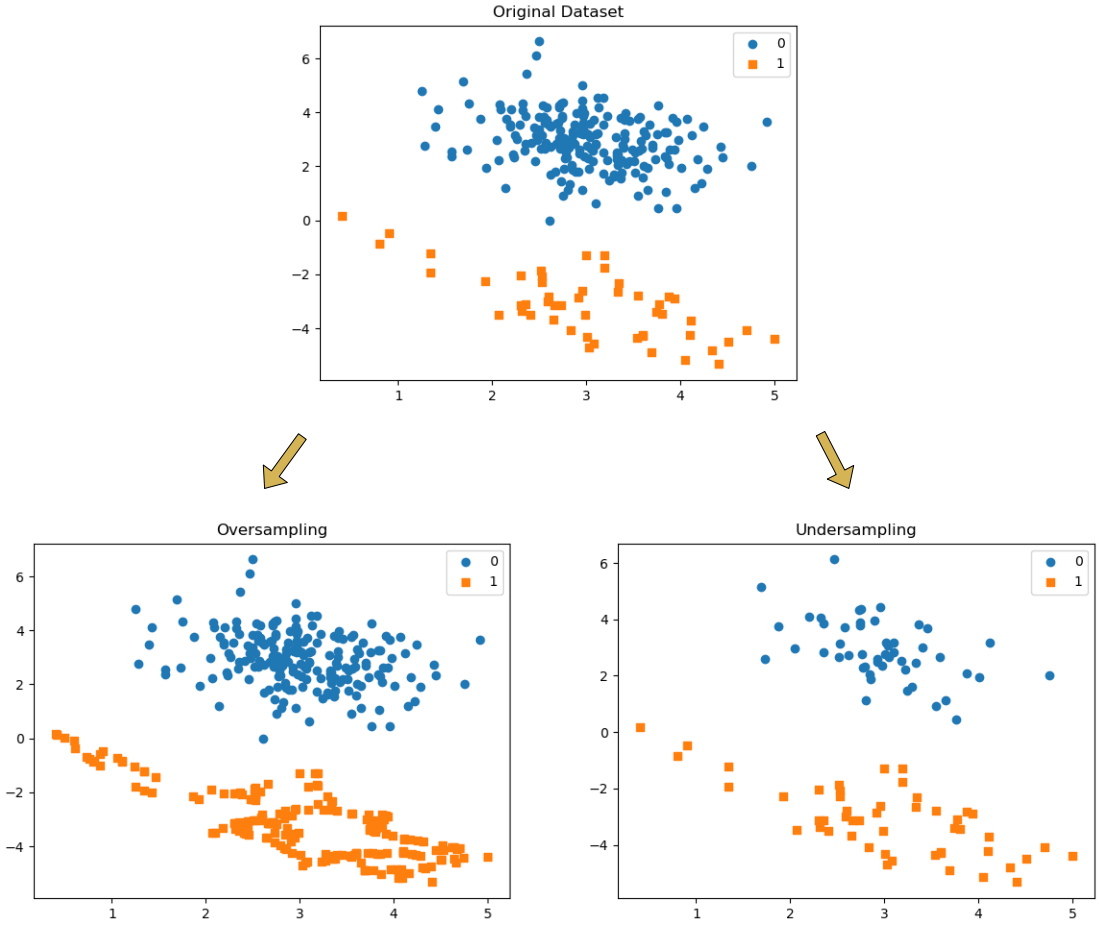


Figure 9 – Oversampling and Undersampling

The algorithmic and cost sensitive approaches will not be further detailed because the focus of this work is the resampling approaches. Section 2.4.2 will detail some examples of algorithms from each of the resampling methods.

2.4.2 Resampling and Algorithms

In this section, some of the most common resampling algorithms in the literature will be presented and detailed. Figure 10 represents the class distribution of a synthetic dataset in a binary classification problem which will be used throughout the examples of each resampling algorithm. In order to make the example easier to understand, the dataset has a total of 100 samples only, as presented in table 3.

Table 3 – Class distribution of the synthetic dataset

Original	
Class	**No. of samples**
0	90
1	10

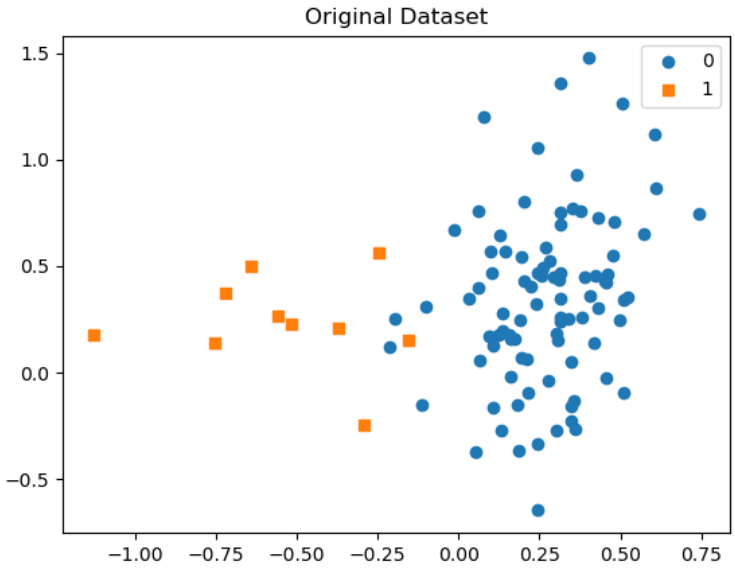


Figure 10 – Synthetic binary classification dataset used in the resampling algorithms examples

2.4.2.1 Random Oversampling (ROS)

The Random Oversampling (ROS) algorithm is a method which uses the random replication of existing minority class samples to tackle the class imbalance issue. The major drawback of this kind of approach to deal with imbalanceness is that it increases the likelihood of overfitting. In a hypothetical classifier, we may be able to construct an adequate model to this dataset, however, as we have replicated new samples, the classifier will probably biased towards these examples (BATISTA; PRATI; MONARD, 2004). For our example, after applying the ROS algorithm would result in the new class distribution presented in table 4.

As mentioned earlier, the new samples created by the ROS algorithm are basically random duplicates of the existing ones. Because of this, if we look at the dataset before and after the resampling is applied, as presented in figure 11, it seems that nothing happened. However, this is not true. The figures look alike because the synthetic examples were replicated from the existing ones.

Table 4 – Class distribution of the synthetic dataset after using the ROS algorithm

ROS	
Class	**No. of samples**
0	90
1	90

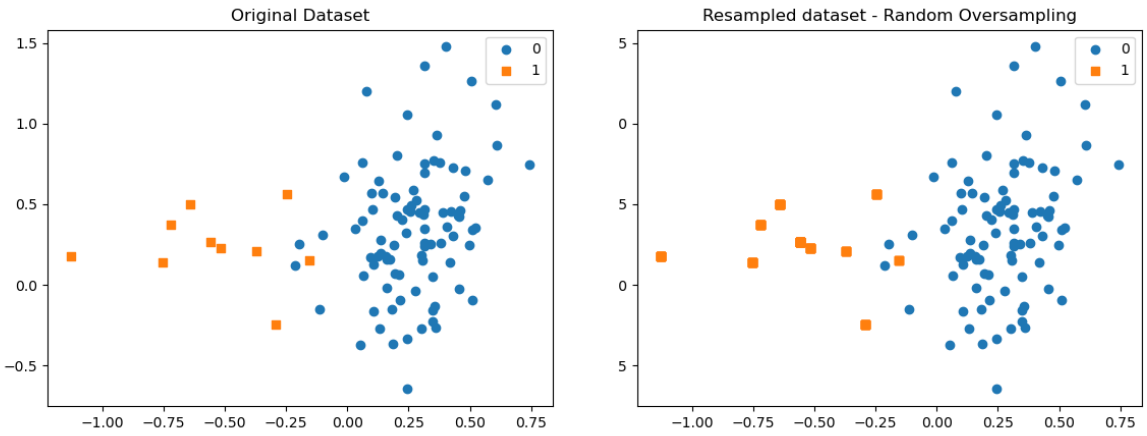


Figure 11 – Class distribution of the synthetic dataset after resampling it with Random Oversampling

2.4.2.2 Synthetic Minority Oversampling Technique (SMOTE)

The SMOTE was initially proposed by (CHAWLA et al., 2002), and its main purpose is to generate new synthetic examples for the minority class, using the existing examples as a starting point.

Basically, the oversampling occurs as it follows: new synthetic examples are introduced for the minority class along the segments that join the k-nearest neighbors of the minority class sample (CHAWLA et al., 2002). This is done through the interpolation of the nearest neighbors, as it follows: the algorithm will calculate the difference between a feature vector of a sample and its nearest neighbor, which basically is the distance between them. Then, it will multiply this value by a random number between 0 and 1, causing it to create a new point along the line that connects the two feature vectors. Figure 12 shows how the new synthetic examples through interpolation.

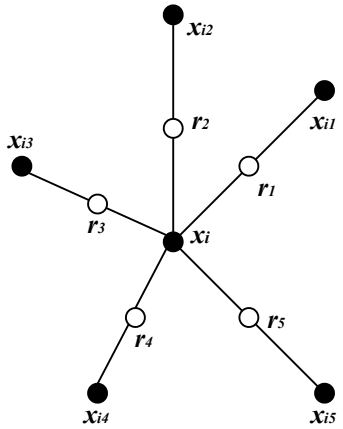


Figure 12 – How the SMOTE synthetic examples are created, for a sample x_i with 5 nearest neighbors.

In figure 12, we see that the example x_i has 5 nearest neighbors, represented by $x_{i1}, x_{i2}, x_{i3}, x_{i4}$ and x_{i5}. The new synthetic examples are represented by r_1, r_2, r_3, r_4 and r_5.

In order to have an example of how the point in between them is calculated, Let us consider a sample $S = (6, 5)$ and its nearest neighbor $N = (4, 3)$ as its nearest neighbor. Let us consider the following notation: f_{ij}, where i represents the sample (1 for S, 2 for N) and j represents the feature number. Thus, for sample S we have f_{11} and f_{12}, while for sample N we have f_{21} and f_{22} We calculate the new sample through the following expression:

$$R_i = S_{ij} + rand(0, 1) * (f_{21} - f_{11}, f_{22} - f_{12}) \tag{2.14}$$

$$R_i = (6, 5) + rand(0, 1) * (6 - 4, 5 - 3) \tag{2.15}$$

$$R_i = (6, 5) + rand(0, 1) * (2, 2) \tag{2.16}$$

The new class distribution after resampling is presented in table 5.

Table 5 – Class distribution of the synthetic dataset after resampling it with SMOTE

SMOTE	
Class	**No. of samples**
0	90
1	90

In figure 13, we can see how the new samples were distributed after applying the resampling algorithm. It is possible to notice that new examples were created in random spots between the existing ones.

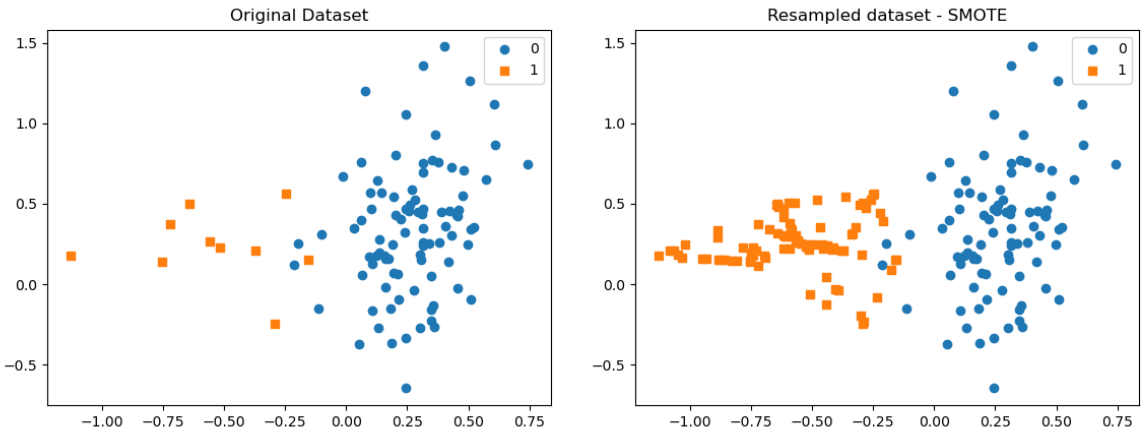


Figure 13 – The difference in the dataset distribution after resampling it with SMOTE

2.4.2.3 Borderline-SMOTE

In the studies of (HAN; WANG; MAO, 2005), the author suggests that the data instances that are far from the borders of the clusters contribute a little for the training process of the classifier. Because of this, the authors propose two new oversampling algorithms called borderline-SMOTE1 and borderline-SMOTE2. The main difference of these two new methods to the other oversampling algorithms is that only the minority class examples at the borderline are oversampled, instead of oversampling all of the examples. These new algorithms are based on the SMOTE oversampling algorithm. The new class distribution is presented in table 6.

The result of the new distribution is also presented in figure 14. As discussed earlier, the Borderline-SMOTE algorithm aims to reinforce the borders of the clusters. Because of this, if we look at the borders of the cluster for class 1 in our example, we can see that a lot more samples were created close to the border. The farther we go away from the border, less new samples were created.

Table 6 – Class distribution of the synthetic dataset after resampling it with Borderline-SMOTE

Borderline-SMOTE	
Class	**No. of samples**
0	90
1	90

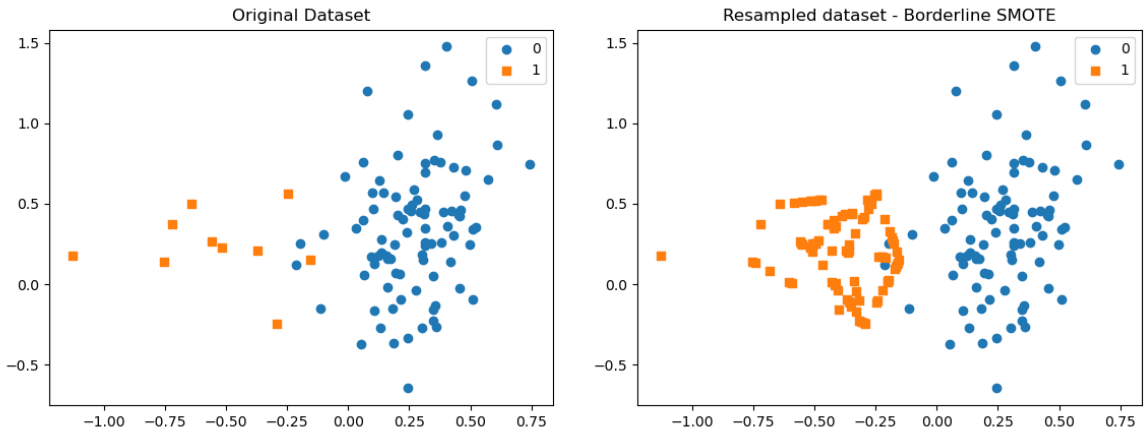


Figure 14 – The difference in the dataset distribution after resampling it with Borderline-SMOTE

2.4.2.4 Adaptive Synthetic Sampling (ADASYN)

The idea behind the ADASYN algorithm is to be an adaptative approach where more synthetic data is generated for the minority class examples that are harder to learn compared to those that are easier to learn. To do this, the algorithm uses a density distribution that measures how difficult it is to learn minority classes examples. This measurement is used in order to decide the number of synthetic examples that will be generated for each minority class example.

As an example, let us consider x_i an example from the set of minority class samples, the number of minority class examples as m_s, the number of nearest neighbors based on the Euclidean distance as K, and Δ_i as the number of majority class examples within the K nearest neighbors of x_i. We then calculate the ratio r_i:

$$r_i = \frac{\Delta_i}{K}, i = 1, 2, ..., m_s \tag{2.17}$$

Then, in order to obtain a density distribution, we normalize the expression 2.17:

$$\hat{r_i} = \frac{r_i}{\sum_{n=1}^{m_s} r_i} \tag{2.18}$$

Greater values of $\hat{r_i}$, represent examples that are more difficult to learn and thus more synthetic examples will be generated in these cases.

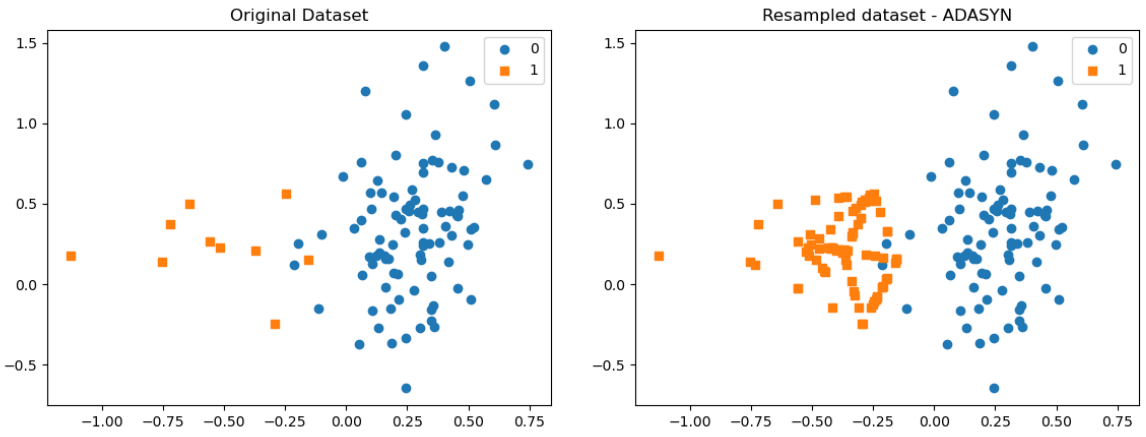
The main objective of doing this is to reduce the bias introduced by class imbalance and also reinforce the classification task towards the difficult examples (HE et al., 2008). Table 7 shows the new class distribution after applying the ADASYN algorithm to the

dataset, and in figure 15 we present the comparison between the original dataset and the new dataset, after applying resampling with ADASYN.

Table 7 – Class distribution of the synthetic dataset after resampling it with ADASYN

ADASYN	
Class	**No. of samples**
0	90
1	90

Figure 15 – The difference in the dataset distribution after resampling it with ADASYN



2.4.2.5 SMOTE-Tomek

The SMOTE-Tomek algorithm is also one of the hybrid approaches to resample data. It is based on the SMOTE algorithm in the oversampling phase, and it also uses the Tomek links algorithm as data cleaning method. Frequently, datasets have class clusters that are not very well defined. In some cases, data examples from the majority class invade the minority class spaces. The opposite is also true, mainly when we introduce new examples for the minority class, which could lead to overfitting when trying to induce a classifier in this situation.

Because of this, the main purpose of the SMOTE-Tomek algorithm is to oversample the minority classes instances using the SMOTE algorithm, and then use a data cleaning method to remove those samples (from both majority and minority classes) that are invading spaces they should not. To do this, the Tomek links method is used as a data cleaning tool.

Tomek Links

In order to be able to understand how the data cleaning method works, we need to define what are the Tomek links, which were originally proposed by (TOMEK et al., 1976). Let us say that we have two samples E_i and E_j from distinct classes. Let's consider the distance between them as $d(E_i, E_j)$. This pair of samples (E_i, E_j) is called a Tomek link if there is not another sample E_l in which $d(E_i, E_l) < d(E_i, E_j)$ or $d(E_j, E_l) < d(E_i, E_j)$. If these examples (E_i, E_j) form a Tomek link, they could be either noise or a borderline. This is a method that may be used to undersample the majority class or as a data cleaning technique, removing the examples that form the Tomek link for both classes (BATISTA; PRATI; MONARD, 2004).

In our example, if we apply the SMOTE-Tomek algorithm to the original dataset, we will have the new distribution as presented in table 8.

Table 8 – Class distribution of the synthetic dataset after resampling it with SMOTE-Tomek

SMOTE-Tomek	
Class	**No. of samples**
0	89
1	89

As mentioned earlier, the first step is applying the SMOTE algorithm to oversample the data. Then we use the Tomek links as a data cleaning technique. In figure 16, we see the original distribution and the result of both steps. First, the SMOTE algorithm was applied and the outcome of this operation is in the bottom left. The two instances of data that are highlighted with the red circle form a Tomek link. Finally, in the second step, they are removed and the final distribution is seen on the bottom right side of figure 16.

2.4.2.6 SMOTE-ENN

The SMOTE-ENN algorithm is a hybrid resampling approach. The main objective of the approach is like the SMOTE-Tomek, which is oversampling the minority classes and applying a data cleaning technique to clean-up the borders of the clusters, to have better defined borders between the classes. This method uses the SMOTE algorithm to oversample the minority class and the Wilson's Edited Nearest Neighbors (Edited Nearest Neighbor (ENN)) (WILSON, 1972) rule as a data cleaning approach. This algorithm promotes a deeper data cleaning than the SMOTE-Tomek, because it tends to remove more samples than the last one.

Edited Nearest Neighbors (ENN)

The ENN rule, proposed by (WILSON, 1972), is a rule that aims to remove any example that have a different class than its three neighbors.For a binary classification

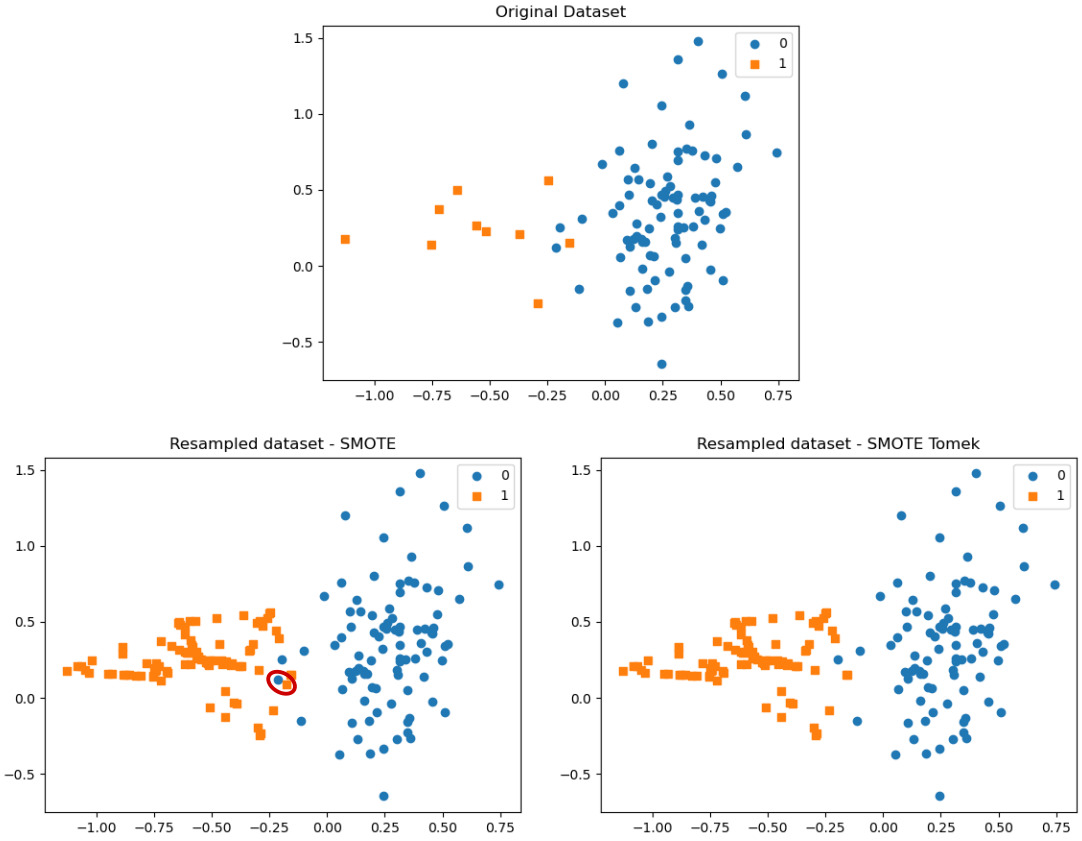


Figure 16 – Sequence of resampling steps done when using the SMOTE-Tomek algorithm. The final result is the image in the bottom right.

problem, let us consider a data sample E_i. If at least two out of three of its neighbors have a different class, then E_i is removed. However, if E_i belongs to the minority class, and at least two out of three neighbors have a different class, then the neighbors that have a different class are removed (BATISTA; PRATI; MONARD, 2004).

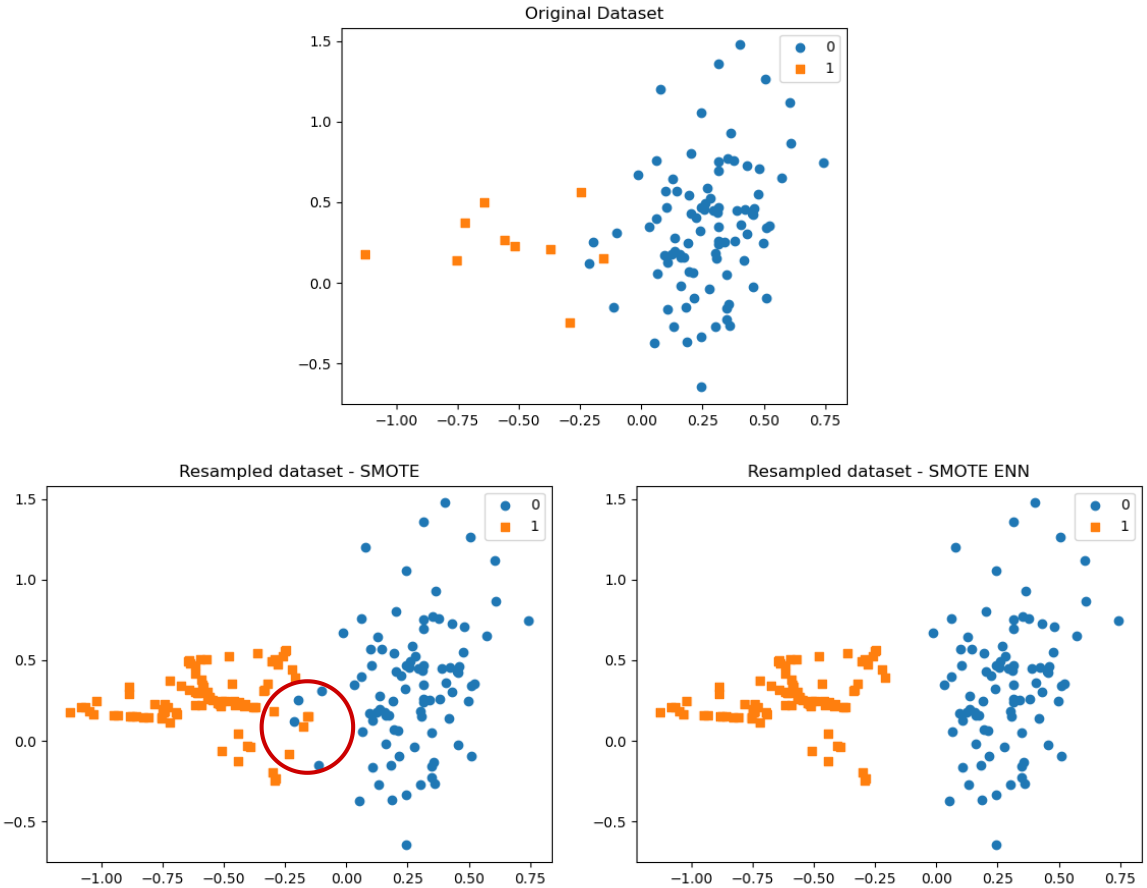
After applying the SMOTE-ENN algorithm to our example, we will have a new class distribution, presented in table 9.

As mentioned before, the SMOTE-ENN algorithm has two phases: the oversampling and the data cleaning through the ENN rule. Figure 17 shows the results for both oversampling and undersampling steps. The image on the top represents the original class distribution. The bottom left one represents the output of the SMOTE algorithm, and the red circle highlights those samples that fit into the ENN rule. The other one on the bottom right, depicts the final result after the data cleaning was applied. We can see that in this case we have a much cleaner borderline, compared to the SMOTE-Tomek approach shown in figure 16. We can see that at the end we have a lot more samples from the minority class and a much cleaner borderline between the two clusters, once they are very well defined.

SMOTE-ENN	
Class	**No. of samples**
0	86
1	85

Table 9 – Class distribution of the synthetic dataset after resampling it with SMOTE-ENN

Figure 17 – Final result comparing the original dataset with the new distribution after applying SMOTE-ENN



3 Related Work

Class imbalanceness is an issue that affects binary, multi-class and hierarchical classification problems. One subject that has been gaining popularity over the past decade is how to handle this class imbalance issue in these different types of classification problems. As explained in section 2.4.1, (FERNÁNDEZ et al., 2013) groups the approaches to deal with imbalanced data in three: data-level, algorithmic level and the cost-sensitive approaches. The current work will focus in the data-level approach, since it is independent of the classification algorithm used. Therefore, the literature review will be focused in those papers where a data-level approach was used as a way to cope with the imbalanced class distributions often found in real-world data.

Our objective in the current work is to investigate in the literature those works where the hierarchical classifier is implemented using a local classification approach, in an imbalanced scenario. It is also of our interest to explore hierarchical classification papers that use data-level techniques to cope with the class imbalance issue. Table 10 shows a list of works published recently which applied either the local classifier in an imbalanced scenario or data-level techniques to cope with class imbalanceness. In table 10, we present which hierarchical classification approach was used, the resampling strategy utilized (oversampling, undersampling or hybrid) and the domain of the datasets used in each work.

From the analysis of the works presented in table 10, we can see that there are some recently published papers that are employing resampling as a strategy to overcome data imbalance. In general, they contemplate a wide range of domains, such as:

- Biology: In (FENG; FU; ZHENG, 2017) (FENG; FU; ZHENG, 2018) for gene ontology prediction.

- Text Categorization: In (RACHMAN; KHODRA; WIDYANTORO, 2018), (KLUNGPORNKUN; VATEEKUL, 2019) and (ADDI; EZZAHIR; MAHMOUDI, 2020). In (RACHMAN; KHODRA; WIDYANTORO, 2018), the classifier is used for sentence categorization in papers, while in (KLUNGPORNKUN; VATEEKUL, 2019) the World Intellectual Property Organization (WIPO) (patents) and Wiki datasets are used. Finally, in (ADDI; EZZAHIR; MAHMOUDI, 2020), the classification task uses texts in Arabic in order to identify sentiments.

- Medical and Medical imaging: In (ABAD; MASLOVE; LEE, 2020) the authors use the classifier to predict discharge of critically ill patients according to variables such

as admission diagnosis and chronic health conditions. In (PEREIRA et al., 2020), the classifier is employed in the COVID-19 identification through CXR images.

- Music: In (PEREIRA; COSTA; SILLA, 2018), the classification task aims to help in music genre recognition.
- Astronomy: In (HOSENIE et al., 2019), the main task is to recognize stars using light curves as features.
- Multiple domains: Some of the related works do not focus in a specific domain to apply any hierarchical classification techniques. In (PEREIRA; COSTA; SILLA, 2021), The local classifiers are tested with datasets from the biological, music, image, and text domains. In (SILVA-PALACIOS; FERRI; RAMÍREZ-QUINTANA, 2017), the authors test a big variety of datasets, which include: medical, text and image.

It is important to emphasize that the datasets used in these works mentioned in table 10 contemplate both single and multi-labeled classification problems. The current work will focus on tree-structured hierarchical classification problems with single path of labels.

Table 10 – Related work published in the literature

Reference	Hierarchical Classification Approach	Single/ Multi-label	Resampling approach	Domain
(FENG; FU; ZHENG, 2017)	LCN	Multi	Over	Biology
(RACHMAN; KHODRA; WIDYANTORO, 2018)	LCPN	Single	Over/ Under	Text
(SILVA-PALACIOS; FERRI; RAMÍREZ-QUINTANA, 2017)	LCPN	Single	Under	Multiple domains
(ABAD; MASLOVE; LEE, 2020)	LCPN	Single	Over/ Hybrid	Medical
(FENG; FU; ZHENG, 2018)	LCN	Multi	Over	Biology
(PEREIRA; COSTA; SILLA, 2018)	Clus-HMC Framework	Multi	Over/ Under	Music
(KLUNGPORNKUN; VATEEKUL, 2019)	Level Based CNN	Multi	Over	Text
(HOSENIE et al., 2019)	LCPN	Single	Under	Astronomy
(PEREIRA et al., 2020)	Clus-HMC Framework	Single	Over/ Under	Medical Imaging
(ADDI; EZZAHIR; MAHMOUDI, 2020)	LCPN	Single	Over	Text
(PEREIRA; COSTA; SILLA, 2021)	LCPN/ LCN/ LCL	Single/ Multi	Over/ Under/ Hybrid	Multiple domains

The related works presented in table 10 make use of the three different resampling approaches mentioned in section 2.4.1. For the oversampling strategy, we may summarize as:

- Oversampling: It is the most widely used technique when it comes to resampling. It was used in (FENG; FU; ZHENG, 2017), (RACHMAN; KHODRA; WIDYANTORO, 2018), (SILVA-PALACIOS; FERRI; RAMÍREZ-QUINTANA, 2017), (ABAD; MASLOVE; LEE, 2020), (FENG; FU; ZHENG, 2018), (PEREIRA; COSTA; SILLA, 2018), (KLUNGPORNKUN; VATEEKUL, 2019), (PEREIRA et al., 2020), (ADDI; EZZAHIR; MAHMOUDI, 2020) and (PEREIRA et al., 2020).
- Undersampling: In our list of related works, it was used by (RACHMAN; KHODRA; WIDYANTORO, 2018), (SILVA-PALACIOS; FERRI; RAMÍREZ-QUINTANA, 2017), (PEREIRA; COSTA; SILLA, 2018), (HOSENIE et al., 2019), and (PEREIRA et al., 2020) and (PEREIRA; COSTA; SILLA, 2021).
- Hybrid: The use of algorithms that combine both oversampling and undersampling techniques was employed in (ABAD; MASLOVE; LEE, 2020) and (PEREIRA; COSTA; SILLA, 2021)

Most of the related work found in the literature that uses resampling also uses local classification approaches. However, some of them used other kind of classifiers. We present below a summary of the hierarchical classification approaches used in each of them.

- LCPN: One of the simplest and most common approaches in the literature. Used in (RACHMAN; KHODRA; WIDYANTORO, 2018), (SILVA-PALACIOS; FERRI; RAMÍREZ-QUINTANA, 2017), (ABAD; MASLOVE; LEE, 2020), (HOSENIE et al., 2019), (PEREIRA; COSTA; SILLA, 2021) and (ADDI; EZZAHIR; MAHMOUDI, 2020).
- LCN: This is also a very common approach in the literature. It was used in (FENG; FU; ZHENG, 2017), (FENG; FU; ZHENG, 2018) and (PEREIRA; COSTA; SILLA, 2021).
- LCL: Among the local classifiers, this approach is the least used. Even though the authors do not mention LCL in (KLUNGPORNKUN; VATEEKUL, 2019), they implement a level based Convolutional Neural Networks (CNN), which means a CNN for each level of the hierarchy, which could be interpreted as a local classifier per level.
- Global: Even though the global classifiers are not the focus of the current work, we included the works of (PEREIRA et al., 2020) and (PEREIRA; COSTA; SILLA, 2018)

because they use one of the state-of-the-art ensembles for hiearchical classification, called Clus-Hierarchical Multi-Label Classification (HMC), which was proposed by (NAKANO; LIETAERT; VENS, 2019). This classifier is combined with resampling techniques in both (PEREIRA et al., 2020) and (PEREIRA; COSTA; SILLA, 2018).

If we would pick the state-of-the-art works for the hierarchical classification in imbalanced scenarios, and which also use data-level approaches to overcome the class imbalanceness issue, they would be (PEREIRA et al., 2020), (ADDI; EZZAHIR; MAHMOUDI, 2020) and (PEREIRA; COSTA; SILLA, 2021). They are the most recent works published in the literature that use multiple resampling techniques. In (ADDI; EZZAHIR; MAHMOUDI, 2020) and (PEREIRA; COSTA; SILLA, 2021), the authors both use local classification approaches. The first uses the LCPN only, while in the second, the author makes use of all kinds of local classifiers. The work of (PEREIRA; COSTA; SILLA, 2021) tests a lot more approaches and different datasets than in (ADDI; EZZAHIR; MAHMOUDI, 2020), and thus we may consider it the one of the best references regarding the hierarchical classification in imbalanced domains.

The other work we mentioned (PEREIRA et al., 2020), focuses in two approaches to resolve the data imbalance issue, which is the data-level by using widely known resampling algorithms as a pre-processing step, and a algorithmic level by using a classifier tailored for hierarchical scenarios called Clus-HMC, proposed by (NAKANO; LIETAERT; VENS, 2019), and is one of the state-of-the-art ensemble algorithms to be used in hierarchical scenarios.

4 Methodology

This chapter describes which steps are taken to develop a hierarchical classifier to be used with imbalanced datasets, and also how to evaluate the performance of these solutions. We may summarize the steps as it follows:

1. Pre-process or clean the data if necessary (section 4.1);
2. Split the data in train and test (section 4.2);
3. Model the hierarchical problem to a suitable representation (4.3);
4. Propose and develop resampling approaches (section 4.4);
5. Model and develop a local hierarchical classifier (section 4.5);
6. Assess the best ways to evaluate the results (section 4.6).

4.1 Datasets and Data pre-processing

In order to evaluate the performance of different resampling approaches and algorithms in the hierarchical classification task, we propose to use multiple datasets for different domains and with different IRs. The hierarchical classification problems chosen so far may be categorized according to the definition by (SILLA; FREITAS, 2011) as:

- Structure to represent the hierarchy: tree;
- Class labels: SPL, which means that each instance of the dataset will have class labels associated with a single path in the hierarchical structure.
- Depth of the data instances: FD, which means that the data instances will have labels associated with a full depth path in the hierarchy.

The list of datasets used in this work is presented in table 11. They were previously used in related works published in the literature. The datasets 1 and 2 (PARMEZAN; SOUZA; BATISTA, 2019) belong to the biological domain, where the first is related to entomology (insects) and the second to ichthyology (fishes), while the dataset 3 (PARMEZAN; SOUZA; BATISTA, 2019) is related to musical instruments classification. The fourth (DIMITROVSKI et al., 2011) is a medical imaging dataset mainly composed by radiographies and the fifth (FELIPE et al., 2019) is called infant Classification of Pain Expressions (iCOPE), and it is composed of audio spectrograms used in the classification of infant's

cry. Lastly, the dataset 6 is a novel one created as part of the this work, and it is a mixture of some images from (PEREIRA et al., 2020) and some from the COVID-19 Image Data Collection (C-19 IDC) (COHEN et al., 2020), which we call RYDLS C-19 IDC. The process to obtain the RYDLS C-19 IDC dataset is further explained in section 4.1.1, once this is a novel dataset that was obtained during the current work. Since the other datasets were already used in previous works in the literature, we do not go over further details of how they were obtained.

Table 11 – Proposed datasets

	Dataset	Domain	Reference
1.	Actinopterygii	Biology	(PARMEZAN; SOUZA; BATISTA, 2019)
2.	Diptera	Biology	(PARMEZAN; SOUZA; BATISTA, 2019)
3.	Instrument	Music	(PARMEZAN; SOUZA; BATISTA, 2019)
4.	ImCLEF07	Medical Imaging	(DIMITROVSKI et al., 2011)
5.	iCOPE	Biology	(FELIPE et al., 2019)
6.	RYDLS C-19 IDC	Medical Imaging	(PEREIRA et al., 2020; COHEN et al., 2020)

The descriptive details for regarding each dataset are presented in the table 12. In this table we show the number of label paths, what is the hierarchical structure that defines the data, what is the maximum depth (levels) of the hierarchy, the number of samples for each level, the total number of samples and the IR, calculated through equation 2.13.

We also present in Appendix A the complete original class distribution of each one of them, in tables 42 through 47 . Their hierarchical class distribution is also presented in Appendix A in tables 48 through 53.

Table 12 – Overview of all the datasets

Dataset	Structure	Label Paths	Label Path Depth	Max. Depth	Samples per Level	Total of Samples	IR
1. Actinopterygii		15		4	22444/22444/22444/22444	22444	149.71
2. Diptera		14		4	21722/21722/21722/17712	21722	12.77
3. Instrument	Tree	31	MLNP	3	9419/9419/9419	9419	41.16
4. ImCLEF07D		26		3	11006/11006/11006	11006	232.33
5. InfantCry		4		2	113/71	113	2.80
6. RYDLS C-19 IDC		10		3	1581/570/570	1581	125.00

4.1.1 RYDLS C-19 IDC Dataset

The C19-IDC (COHEN et al., 2020) dataset is an ongoing project of a public dataset that contains CXR and Computed Tomography (CT) scan images from public sources as well as through indirect collection from hospitals and physicians. It is a dataset which is constantly being updated with images of lungs which were infected either by COVID-19 or other types of pneumonia. However, it doesn't contain any images of healthy

lungs. Even though the C-19 IDC dataset (COHEN et al., 2020) does not contain any images of healthy lungs, RYDLS (PEREIRA et al., 2020) has them, making these images suitable to complement the up-to-date images of infected lungs obtained in C-19 IDC.

Having in mind that RYDLS (PEREIRA et al., 2020) contains healthy images and that C-19 IDC (COHEN et al., 2020) contains very up-to-date images of infected lungs, we propose to create a new dataset which is a mixture of both, using the healthy images from RYDLS with the infected ones of C-19 IDC, and which is hereafter referred as the RYDLS C-19 IDC dataset.

In the images that came from the RYDLS dataset, no data cleaning operations were necessary, but in the C-19 IDC (COHEN et al., 2020) some cleaning operations were applied. The main reason for this is that the RYDLS (PEREIRA et al., 2020) contains only CXR images, thus all of the CT images from C-19 IDC were removed.

After merging both datasets, the image features were extracted using the Local Binary Patterns (LBP) image descriptor, using 8 neighboring pixels and a radius of 2 with the non-rotation invariant method. This descriptor was chosen due to its reduced computational complexity compared to the most recent ones, like CNNs. The result of the extraction was a feature matrix, where a filter to remove the classes with less than 7 samples was applied, in order to avoid issues when running resampling with SMOTE based resamplers with a number of neighbors k = 5. The final class distribution and the filtered classes of the RYDLS C-19 IDC dataset is presented in table 13. The final distribution without the filtered classes is also presented in table 47.

4.2 Cross-validation

To reduce the effect of testing only one split of the data like in the hold-out strategy, all of the experiments in this work use a stratified k-fold validation with k = 5 to split the training and test sets. The main reason for using the stratified version of it is to keep the same proportions in both training and testing datasets, compared to the original dataset (KOHAVI, 1995).

In order to have a easy to follow example of how the methodology works, let us take the class distribution from the sample dataset presented in table 1. If we take the example shown in table 1 and use the stratified cross-validation with 5 folds, we will have the data divided in train and test folds as presented in table 14.

The fold 1 on table 14 will be used throughout the examples in the subsequent sections. Considering that we propose to use the LCPN and the LCN approaches in the current work, tables 15 and 16 show us the class distribution of fold 1 from table 14 in the LCPN and LCN scenarios, respectively. The IR in both tables was calculated with

Table 13 – Final class distribution of the RYDLS C-19 IDC dataset showing the classes that were filtered.

	Class	count
Final distribution	R/Normal	1000
	R/Pneumonia/Viral/COVID-19	476
	R/Pneumonia/Fungal/Pneumocystis	24
	R/Pneumonia/Bacterial/Streptococcus	18
	R/Pneumonia/Viral/SARS	16
	R/Tuberculosis	11
	R/Pneumonia/Viral/MERS-CoV	10
	R/Pneumonia/Bacterial/Legionella	9
	R/Pneumonia/Bacterial/Klebsiella	9
	R/Pneumonia/Lipoid	8
Filtered (removed)	R/Pneumonia/Bacterial/Mycoplasma	6
	R/Pneumonia/Viral/Varicella	5
	R/Pneumonia/Bacterial/E.Coli	4
	R/Pneumonia/Bacterial/Nocardia	4
	R/Pneumonia/Viral/Herpes	3
	R/Pneumonia/Viral/Influenza	3
	R/Pneumonia/Bacterial	2
	R/Pneumonia/Fungal/Aspergillosis	2
	R/Pneumonia/Viral/Influenza/H1N1	1
	R/Pneumonia/Bacterial/Staphylococcus/MRSA	1
	R/Pneumonia/Aspiration	1
	R/Pneumonia/Bacterial/Chlamydophila	1

Table 14 – Class distribution of the sample dataset after the stratified k-fold split with k = 5

	Fold 1		Fold 2		Fold 3		Fold 4		Fold 5	
Class Label	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**
R/Normal	800	200	800	200	800	200	800	200	800	200
R/Viral/COVID-19	383	96	383	96	383	96	384	95	383	96
R/Fungal/Pneumocystis	24	6	24	6	24	6	24	6	24	6
R/Bacterial/Streptococcus	17	4	18	4	18	4	18	4	17	5
R/Viral/SARS	13	3	13	3	13	3	12	4	13	3
R/Lipoid/Non applicable	11	3	10	3	10	3	10	3	11	2
R/Bacterial/Mycoplasma	9	2	9	2	9	2	8	3	9	2
R/Bacterial/Klebsiella	8	2	8	2	8	2	8	2	8	2
R/Bacterial/Legionella	6	2	6	2	6	2	7	1	7	1

equation 2.13.

4.3 Taxonomy

Adapting the inherent hierarchy for each one of the datasets and modeling it in an algorithmic representation is one of the core steps of the process to build a hierarchical classifier. This structure is basically responsible to dictate how the classes relate to each

Table 15 – Class distribution for the LCPN scenario of the data from table 14, after the stratified k-fold split

Class Label	IR	No. Samples (Training)	No. samples (Testing)
Parent Node: R/ (root)			
R/Normal	72.73	800	200
R/Viral		396	99
R/Bacterial		40	11
R/Fungal		24	6
R/Lipoid		11	2
Parent: Node R/Bacterial			
R/Bacterial/Streptococcus	2.83	17	5
R/Bacterial/Mycoplasma		9	2
R/Bacterial/Klebsiella		8	2
R/Bacterial/Legionella		6	2
Parent: Node R/Viral			
R/Viral/COVID-19	29.46	383	96
R/Viral/SARS		13	3

Table 16 – Class distribution for the LCN scenario of the of the data from table 14, after the stratified k-fold split

Class label	IR	No. samples (Training)	No. samples (Testing)
R/Normal	0.59	800	200
Other		471	118
R/Lipoid	114.55	11	3
Other		1260	315
R/Fungal	51.96	24	6
Other		1247	312
R/Bacterial	30.78	40	10
Other		1231	308
R/Viral	2.21	396	99
Other		875	219
R/Bacterial/Streptococcus	1.35	17	4
Other		23	6
R/Bacterial/Klebsiella	4.0	8	2
Other		32	8
R/Bacterial/Mycoplasma	3.44	9	2
Other		31	8
R/Bacterial/Legionella	5.67	6	2
Other		34	8
R/Viral/COVID-19	0.0034	383	96
Other		13	3
R/Viral/SARS	29.46	13	3
Other		383	96

other.

Due to the nature of the datasets that were chosen, as presented in table 12, which are all classified in the SPL category, the data structure that is most suitable to define the taxonomy used in the hierarchical classifier is a tree. In this specific case, a generic tree, where a parent node may have one or more child nodes.

To better illustrate how the taxonomic structure is assembled, let us use the class labels and distribution from fold 1 in table 14. Considering all the labels defined in the aforementioned table, the R represents the root of the tree and the slash symbols ('/') separate a parent class from its sub-class. For example, for the class R/Viral/COVID-19:

- R: It is a node that represents the root of the tree.
- Viral: It represents a child node of R.
- COVID-19: It represents a child node of the node R/Viral, since COVID-19 is a viral type of pneumonia.

Thus, if we generalize the same logic to all labels from table 14, that will generate the taxonomy depicted in figure 18.

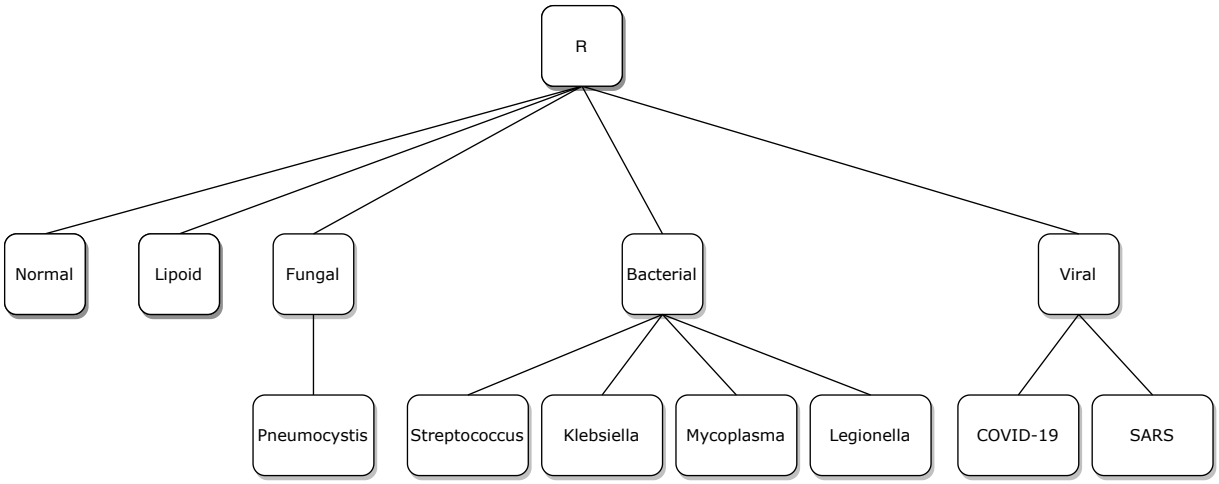


Figure 18 – Taxonomy produced from the class labels in table 14

Each node of the tree represents a data class. Also, each one of them will store the training data for its respective class, depending on the data policy chosen to be used in the hierarchical classifier.

4.3.1 Data retrieval

In hierarchical classification tasks that use a local classification approach, the positive classes for each target class are retrieved according to a policy. There are multiple

policies, which were already detailed in section 2.1. In the next few examples, we will use the siblings policy.

Even though we are using the same policy in the examples for both local classifiers, the way the data is retrieved for the LCPN and LCN scenarios is slightly different because of some adaptations that needed to be done. Both will be explained in more depth in subsections 4.3.1.1 and 4.3.1.2.

4.3.1.1 LCPN Scenario

In the LCPN scenario, the key idea is that a parent node needs to be capable to distinguish data from any of its child (or descending) nodes, which would make it a multi-class classifier. Thus, the positive classes for a particular node are the descending classes for this node and its own class, if the problem is a non-mandatory leaf node prediction.

Let us consider the taxonomy from figure 18. If we take R/Viral parent node as an example and retrieve the positive classes for it according to the siblings policy, defined in section 2.3.1, they would be: R/Viral/COVID-19, R/Viral/SARS and the R/Viral class itself, if the problem is non-mandatory leaf node prediction. Because of this, the R/Viral node would have as positive data from the COVID-19 and SARS child nodes.

Another important part of the data retrieval is that all the descending classes from a particular child node are relabeled to its parent class. In figure 18, if we were trying to retrieve data for the root (R) node, we would have two levels until we reached the leaf nodes. The classes that are not immediate child nodes of the root (R) class would need to be relabeled. In this case, the data from classes COVID-19 and SARS would be relabeled as R/Viral, for example. Table 17 shows the result of relabeling the data for the root (R) node.

Table 17 – Relabeling of the data in the LCPN scenario for the R (root) node

Parent	Class label	Relabeled as
R (Root)	R/Normal	not relabeled
	R/Viral	not relabeled
	R/Viral/COVID-19	R/Viral
	R/Viral/SARS	R/Viral
	R/Bacterial	not relabeled
	R/Bacterial/Streptococcus	R/Bacterial
	R/Bacterial/Mycoplasma	R/Bacterial
	R/Bacterial/Klebsiella	R/Bacterial
	R/Bacterial/Legionella	R/Bacterial
	R/Fungal	not relabeled
	R/Fungal/Pneumocystis	R/Fungal
	R/Lipoid	not relabeled

4.3.1.2 LCN Scenario

In the LCN scenario, the main idea is that a specific node needs to be able to distinguish data between any of its positive classes (its own class and descending nodes) and the negatives ones. This means that the LCN approach utilizes one binary classifier in each node of the hierarchy, instead of using multi-class classifiers as in the LCPN approach.

Again, let us use the figure 18 as an example. If the chosen node is R/Viral, then the positive classes for this one are: R/Viral/COVID-19 and R/Viral/SARS. However, we need to retrieve the negative classes as well. They are all the other classes that are derived from the same parent as R/Viral, which means that they would be R/Normal, R/Fungal, R/Lipoid, R/Bacterial and all its respective sub-classes. The table 18 presents which ones are the positive and negative classes for the R/Viral node.

Table 18 – Positive and negative classes in the LCN scenario for the R/Viral node

Node	Positive classes	Negative classes
R/Viral	R/Viral	R/Normal
	R/Viral/COVID-19	R/Bacterial
	R/Viral/SARS	R/Bacterial/Streptococcus
		R/Bacterial/Mycoplasma
		R/Bacterial/Klebsiella
		R/Bacterial/Legionella
		R/Fungal
		R/Fungal/Pneumocystis
		R/Lipoid

In the same way we need to relabel the data in the parent nodes of the LCPN approach, we also need to do it for each node of the LCN strategy as well. However, in this case the relabeling works slightly different. Here the positive classes (including descending ones) will be relabeled to the current node's class and the negative classes will be relabeled to 'Other'. If we use the R/Viral class as an example, all of its descending classes (COVID-19 and SARS) would be relabeled to R/Viral only. Also, all the samples from the negative classes, like R/Normal or R/Bacterial/Streptococcus would be relabeled to 'Other'. The result of the relabeling is presented in table 19.

Table 19 – Relabeling of the data in the LCN scenario for the R (root) node

Node	Class label	Relabeled as
R/Viral	R/Viral	not relabeled
	R/Viral/COVID-19	R/Viral
	R/Viral/SARS	R/Viral
	R/Normal	Other
	R/Bacterial	Other
	R/Bacterial/Streptococcus	Other
	R/Bacterial/Mycoplasma	Other
	R/Bacterial/Klebsiella	Other
	R/Bacterial/Legionella	Other
	R/Fungal	Other
	R/Fungal/Pneumocystis	Other
	R/Lipoid	Other

4.4 Data Resampling

In order to evaluate the effectiveness of using data resampling algorithms in the imbalanced datasets, first we need to establish a baseline for comparison. Because of this, the first experiments that will be performed are those without any approaches to deal with class imbalanceness.

In addition to establishing a baseline, multiple resampling approaches will be implemented and tested. In this work, we propose four approaches: flat resampling, local resampling, threshold selective resampling and local selective resampling. These approaches will be further detailed in sub-sections 4.4.1, 4.4.2, 4.4.3 and 4.4.4. It is important to remember that first three resampling approaches (flat, local and threshold selective) are executed as a data pre-processing step, while the last one (local selective) is executed during the training process, which means that data is resampled during the training procedure.

To give a concrete example of how the four approaches work, let us consider data distribution from fold 1 in table 14. Before resampling the dataset, according to what was defined in sections 4.2 and 4.3, the data will be split in 5 folds using the stratified k-fold cross-validation and then the tree-like structure shown in figure 18 will be assembled. An example of the class distribution after the k-fold split is presented in tables 14, 15 and 16. The distributions presented in tables 15 and 16 are the ones used for the LCPN and LCN scenarios. These number of samples will be used throughout this entire section to explain in a more practical way how each approach works.

4.4.1 Flat resampling

The simplest and most used approach to resample data in the literature is the one that we will refer to as flat resampling. It consists of applying the algorithms before training, without considering any local class distributions. Let's use as an example the training folds distribution from table 14.

When using this strategy, the training data will be resampled with the selected algorithm before the training phase and that will result in the new class distribution showed in table 20. For this example, we used the SMOTE algorithm.

Table 20 – New class distribution of the of the data from table 14, after the flat resample approach was performed with the SMOTE algorithm

Class Label	**No. samples before SMOTE (Training)**	**No. samples after SMOTE (Training)**	**No. samples (Testing)**
R/Normal	800	800	200
R/Viral/COVID-19	383	800	96
R/Fungal/Pneumocystis	24	800	6
R/Bacterial/Streptococcus	17	800	4
R/Viral/SARS	13	800	3
R/Lipoid/Non applicable	11	800	3
R/Bacterial/Mycoplasma	9	800	2
R/Bacterial/Klebsiella	8	800	2
R/Bacterial/Legionella	6	800	2

After the resampling is done, the new data is saved in separate Comma Separated Values (CSV) files for each local classifier . The summary of the steps taken during the resampling procedure are presented in algorithm 1. The process is also summarized by figure 19, which illustrates all the steps from the algorithm. The terminology used in the algorithm definitions is presented below. This terminology is used in the algorithm definitions of sections 4.4.2, 4.4.3 and 4.4.4 as well.

- $Tree$: Represents the tree of local classifiers;
- Lc_i: They represent the i^{th} local classifier object, composed by $data$ (the subset of data of the given local classifier) and $lblPath$ (the label path for the given local classifier);
- D: Represents the original dataset;
- CV: Represents the the list of Cross Validation splits;
- Tr: Training folds of data;
- Ts: Test fold of data.

Algorithm 1 Flat Resampling

$Tree \leftarrow [Lc_1, Lc_2, ...Lc_n]$ ▷ Set of Local Classifiers
$D \leftarrow loadData()$ ▷ 1. Load original dataset
$CV \leftarrow StratifiedCrossValidation(k = 5)$ ▷ 2. Split data in train/test
for each Tr, Ts in CV **do** ▷ 3. Iterate over the 5 different splits
  $Tr' \leftarrow resample(Tr)$ ▷ 4. Resample data
  for each Lc in $Tree$ **do** ▷ 5. Distribute data in the local classifiers
    $Lc.data \leftarrow findSubset(Tr', Lc.lblPath)$ ▷ 6. Find local classifier subset of data
    $saveFile(Lc)$ ▷ 7. Save Data
  end for
  $saveFile(Ts)$ ▷ 7. Save Data
end for

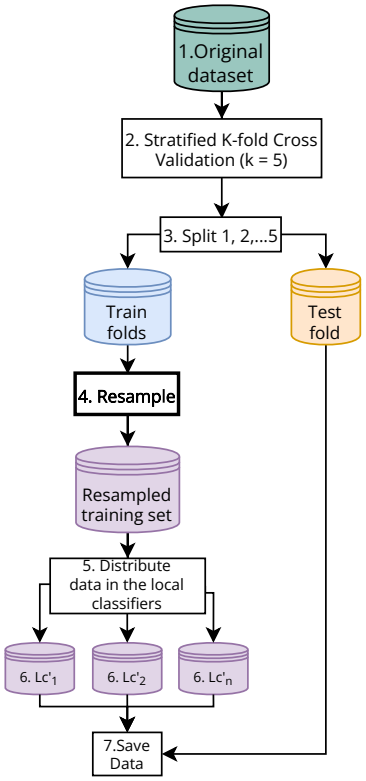


Figure 19 – Flat resampling approach flowchart

4.4.2 Local resampling

Another strategy proposed in this work to overcome the imbalanced class distribution is the one here denominated as local resampling. It consists of applying the algorithms to perform data balancing for each local classifier of the hierarchical structure defined in section 4.3. With this strategy, we aim to make the class distribution of each local classifier balanced. In this approach, the class hierarchy is considered when applying the data resampling algorithms. Again, let's use the taxonomy from figure 18 with the distribution in table 14 as an example.

If we use the LCPN classification approach, we will have multi-class classifiers in each parent node of the hierarchy, which in this case are the following ones: R, R/Bacterial and R/Viral. The local resampling approach will apply the resampling algorithm in these nodes. Table 21 presents the new distribution after resampling in these parent nodes.

Table 21 – New class distribution of the data from table 15 in the LCPN scenario after the local resample approach was performed with the SMOTE algorithm

Class Label	**No. Samples before SMOTE (Training)**	**No. samples after SMOTE (Training)**	**No. samples (Testing)**
Parent Node: R/ (root)			
R/Normal	800	800	200
R/Viral	396	800	99
R/Bacterial	40	800	11
R/Fungal	24	800	6
R/Lipoid	11	800	2
Parent: Node R/Bacterial			
R/Bacterial/Streptococcus	17	17	5
R/Bacterial/Mycoplasma	9	17	2
R/Bacterial/Klebsiella	8	17	2
R/Bacterial/Legionella	6	17	2
Parent: Node R/Viral			
R/Viral/COVID-19	383	383	96
R/Viral/SARS	13	383	3

However, if the LCN classification approach is applied, we will have binary classifiers for each node presented in the figure 18, with the exception of the root and the R/Fungal/Pneumocystis nodes. So, except for these two nodes, all of the other nodes will face data resampling. To exemplify this approach, let us consider the class distribution in table 16. After the resampling is performed in all nodes, the new distribution will look like the one in table 22.

After resampling the data, we save the data for each of the local classifiers in a separate CSV file. We present in algorithm 2 a list of steps that summarize the Local resampling procedure. Figure 20 also depicts the steps shown in the algorithm.

Table 22 – New class distribution of the data from table 16 in the LCN scenario after the local resample approach was performed with the SMOTE algorithm

Class label	IR	No. samples before SMOTE (Training)	No. samples after SMOTE (Training)	No. samples (Testing)
R/Normal	0.59	800	800	200
Other		471	800	118
R/Lipoid	114.55	11	1260	3
Other		1260	1260	315
R/Fungal	51.96	24	1247	6
Other		1247	1247	312
R/Bacterial	30.78	40	1231	10
Other		1231	1231	308
R/Viral	2.21	396	875	99
Other		875	875	219
R/Bacterial/Streptococcus	1.35	17	23	4
Other		23	23	6
R/Bacterial/Klebsiella	4.0	8	32	2
Other		32	32	8
R/Bacterial/Mycoplasma	3.44	9	31	2
Other		31	31	8
R/Bacterial/Legionella	5.67	6	34	2
Other		34	34	8
R/Viral/COVID-19	0.0034	383	383	96
Other		13	383	3
R/Viral/SARS	29.46	13	383	3
Other		383	383	96

Algorithm 2 Local Resampling

$Tree \leftarrow [Lc_1, Lc_2, ...Lc_n]$ ▷ Set of Local Classifiers
$D \leftarrow loadData()$ ▷ 1. Load original dataset
$CV \leftarrow StratifiedCrossValidation(k = 5)$ ▷ 2. Split data in train/test
for each Tr, Ts in CV **do** ▷ 3. Iterate over the 5 different splits
  for each Lc in $Tree$ **do** ▷ 4. Distribute data in the local classifiers
    $Lc.data \leftarrow findSubset(Tr, Lc.lblPath)$ ▷ 5. Find local classifier subset of data
    $Lc' \leftarrow resample(Lc)$ ▷ 6. Resample data
    $saveData(Lc')$ ▷ 7. Save Data
  end for
  $saveData(Tr)$ ▷ 7. Save Data
end for

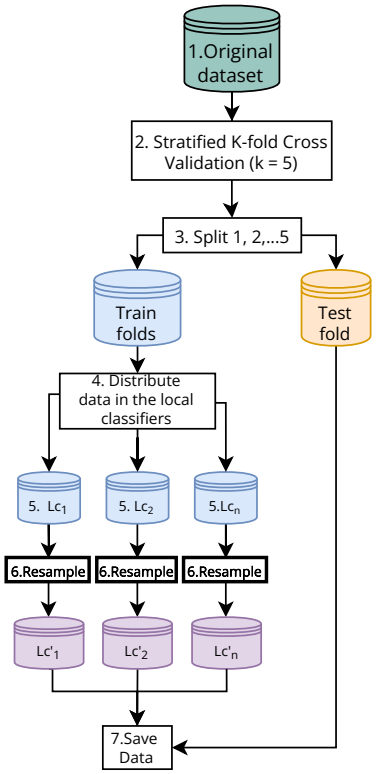


Figure 20 – Local resampling approach flowchart

4.4.3 Threshold selective resampling

This approach is an extension of the local resampling approach (section 4.4.2). The only difference here is that the nodes which will go through data resampling are chosen according to a criterion. We propose to use as a selection criterion a metric called Imbalance Ratio, which is defined in (ORRIOLS-PUIG; BERNADÓ-MANSILLA, 2009) as:

$$IR = N_{majority}/N_{minority} \tag{4.1}$$

Basically, we calculate the IR of a node by computing how many samples there are in the majority class, dividing it by the amount of samples of the minority class. If the IR is greater than 1, that means there are more samples from the majority class than the minority one. One important reminder is that we considered this approach to be more suitable for the LCN scenario, once they use binary classifiers for each node of the hierarchy with the objective of predicting a data class as belonging to the positive or negative class. The objective of this approach is to resample the binary classifiers where the positive class is a minority class and do not resample the binary local classifiers where the positive class is a majority class, in order to avoid negatively affecting the classification

performance for the majority class. Because of this, we change equation 4.2 to:

$$IR = N_{negative}/N_{positive} \tag{4.2}$$

With equation 4.2, we are trying to find out if the negative class(es) for a particular node outnumber the positive class. If so, that means $IR > 1.0$, then we need to resample the data for that node. Otherwise, if $IR \leq 1.0$, we don't need to resample the data to overcome the imbalance towards the negative class. The value $IR = 1.0$ acts like the threshold that will either enable or disable the resampling to be executed, and that's why we named the approach as Threshold Selective Resampling.

Let us use the class distribution of table 16 and the taxonomy from figure 18 as an example to show how this will be performed. As stated before, only those nodes with $IR > 1.0$ will go through resampling. From table 16, we can see that for the R/Normal and R/Viral/COVID-19 classes, this condition is not true. Thus, these nodes will not have their data resampled. The final distribution after applying the Threshold Selective resampling with the SMOTE algorithm is shown in table 23.

Table 23 – Class Distribution of LCN scenario after the threshold selective approach was performed with the SMOTE algorithm

Class label	IR	No. samples before SMOTE (Training)	No. samples after SMOTE (Training	No. samples (Testing)
R/Normal	0.59	800	800	200
Other		471	471	118
R/Lipoid	114.55	11	1260	3
Other		1260	1260	315
R/Fungal	51.96	24	1247	6
Other		1247	1247	312
R/Bacterial	30.78	40	1231	10
Other		1231	1231	308
R/Viral	2.21	396	875	99
Other		875	875	219
R/Bacterial/Streptococcus	1.35	17	23	4
Other		23	23	6
R/Bacterial/Klebsiella	4.0	8	32	2
Other		32	32	8
R/Bacterial/Mycoplasma	3.44	9	31	2
Other		31	31	8
R/Bacterial/Legionella	5.67	6	34	2
Other		34	34	8
R/Viral/COVID-19	0.0034	383	383	96
Other		13	13	3
R/Viral/SARS	29.46	13	383	3
Other		383	383	96

After applying the local resampling in all local classifiers that have an IR that is bigger than the threshold of $IR > 1.0$, we save the data for each node in a separate CSV

file. In algorithm 3 we present the summary of steps executed for this approach. In figure 21 we also illustrate these steps.

Algorithm 3 Threshold Selective Resampling

```
Tree ← [Lc_1, Lc_2, ...Lc_n]                          ▷ Set of Local Classifiers
D ← loadData()                                        ▷ 1. Load original dataset
CV ← StratifiedCrossValidation(D, k = 5)              ▷ 2. Split data in train/test
for each Tr, Ts in CV do                              ▷ 3. Iterate over the 5 different splits
    for each Lc in Tree do                            ▷ 4. Distribute data in the local classifiers
        Lc.data ← findSubset(Tr', Lc.lblPath)         ▷ 5. Find local classifier subset of data
        IR ← calculateIR(Lc)                          ▷ 6. Calculate IR
        if IR > 1.0 then                              ▷ 7. IR exceeds threshold?
            Lc' ← resample(Lc)                        ▷ 8. Resample data
            saveData(Lc')                             ▷ 9. Save Data
        else
            saveData(Lc)                              ▷ 9. Save Data
        end if
    end for
    saveData(Tr)                                      ▷ 9. Save Data
end for
```

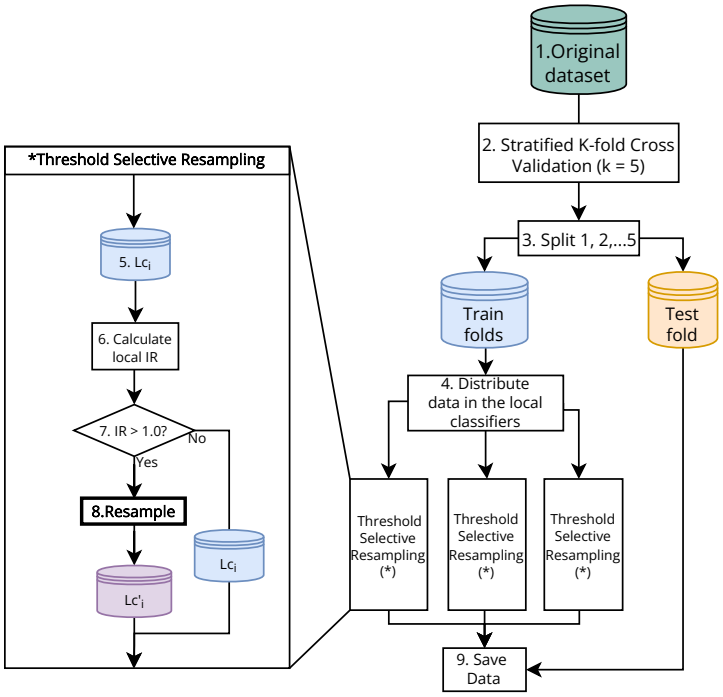


Figure 21 – Threshold Selective resampling approach summarized

4.4.4 Local Selective Resampling

In order to understand this approach, we need to consider the data distribution across the local classifiers. Let us use the LCPN scenario as an example, considering the

distribution from table 15 and the tree from figure 4. In this scenario, as seen in table 15, we will have three parent node classifiers (In the nodes R, R/Bacterial and R/Viral). Considering the data retrieval process described in section 4.3.1, we will then have three subsets of data, one for each of these nodes.

The key behind the local selective resampling method is that we will be selecting the most suitable resampling algorithm for each of the local subsets of each local classifier. To do this, we will split each local subset in sub-train and validation using the stratified hold-out technique, with 80% of the samples in the sub-train set and 20% in the validation subset.

After the data is splitted, this sub-train set is resampled with a resampling algorithm. Then, the local classifier is trained with the resampled set and validated with the validation set. We test all of the resampling algorithms mentioned in section 4.4.5, and also without any resampling algorithm. The Macro Averaged F1-Score for each of the tests is evaluated, and then the classifier with the best performance is saved. The steps taken to execute this procedure are summarized in algorithms 4 and 5, in two separate functions, one controlling the local selective resampling main flow and another that executes the resampler selection. These steps are also illustrated by figure 22. In addition to the terminology defined in section 4.4.1, we also define new variables in the algorithms definition:

- R: List of resampling algorithms from section 4.4.5;
- clf: Best classifier obtained from the resampler selection procedure;
- $best$: Variable that stores the best Macro-Avg. F1 Score;
- $bestClf$: Variable that stores the classifier with the best score;
- STr: Sub-train set, obtained after splitting the local classifier data in sub-train and validation;
- STr': Sub-train set after resampling;
- Vs: Validation set, obtained after splitting the local classifier data in sub-train and validation.

Let us consider the distribution in table 15. An example of a result of applying the Local Selective Resampling strategy to a dataset is presented in table 24. In this table we have the Macro-Avg F1 scores for each resampling algorithm tested in each of the local classifiers.

In this example, the local classifier in node R performed better with the Borderline-SMOTE resampling algorithm, the local classifier in node R/Bacterial performed better

Algorithm 4 Local Selective Resampling

```
Tree ← [Lc_1, Lc_2, ...Lc_n]                          ▷ Set of Local Classifiers
R ← [None, ROS, SMOTE, ...]                           ▷ Set of Resamplers
D ← loadData()                                        ▷ 1. Load original dataset
CV ← StratifiedCrossValidation(k = 5)                 ▷ 2. Split data in train/test
for each Tr, Ts in CV do                              ▷ 3. Iterate over the 5 different splits
    for each Lc in Tree do                            ▷ 4. Distribute data in the local classifiers
        clf ←resamplerSelection(Tr, Lc, R)            ▷ 5. Call resampler selection
        save(Tree, clf)
    end for
    predicted ← predict(Ts, Tree)                     ▷ 6. Predict using the test fold
    calculateMetrics(predicted)                       ▷ 7. Calculate the metrics
end for
```

Algorithm 5 Resampler Selection function

```
best ← 0.0
bestClf ← null
Lc.data ← findSubset(T_r, Lc.lblPath)                 ▷ 5.1. Find local classifier subset of data
STr, Vs ← StratifiedHoldout(Lc)                       ▷ 5.2. Split data in train(80%)/test(20%)
for each resampler in R do                            ▷ 5.3. Test all resampling algorithms
    STr' ← resample(STr)                              ▷ 5.4. Resample data
    clf ← train(STr')                                 ▷ 5.5. Train local classifier
    predictions ← predict(Vs, clf)                    ▷ 5.6. Predict using the validation set
    result ← calculateMetrics(predictions)            ▷ 5.7. Calculate the metrics
    if result > best then                             ▷ 5.8. Check if it was the best result
        bestClf ← clf                                 ▷ 5.9. Save the best classifier
    end if
end for
return bestClf
```

Table 24 – Local Selective Resampling result example

Local Classifier	None	ROS	SMOTE	Borderline-SMOTE	ADASYN	SMOTE-ENN	SMOTE-Tomek	Best Resampler
R	0.619	0.626	0.620	**0.629**	0.626	0.614	0.620	Borderline-SMOTE
R/Pneumonia/Bacterial	**0.390**	0.243	0.217	0.242	0.223	0.198	0.217	None
R/Pneumonia/Viral	0.522	0.656	**0.660**	0.656	0.433	0.656	0.656	SMOTE

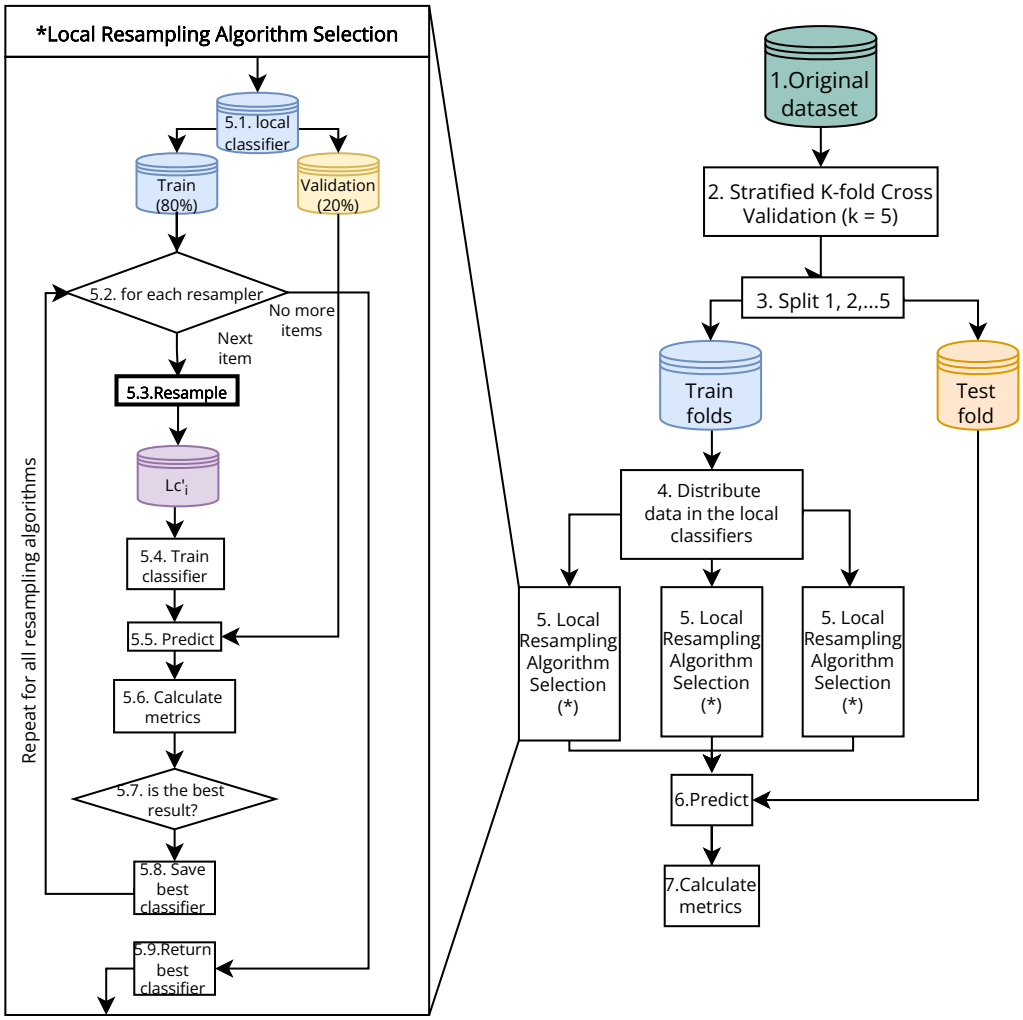


Figure 22 – Local Selective Resampling approach summarized

without resampling, and the R/Viral one performed better with the SMOTE resampler. In this scenario, we save the local classifier in R that was trained with a subset of data that was resampled using Borderline-SMOTE, we save the local classifier for the R/Bacterial node that was trained without resampling and finally, the classifier for the R/Viral node that was trained with the dataset resampled with SMOTE. These local classifiers are used in the final prediction, where other unseen samples are used for the validation of the hierarchical classifier.

4.4.5 Resampling algorithms

For all the three approaches, we propose to use multiple resampling algorithms of different types. The table 25 summarizes the algorithms used to resample the data. As mentioned in section 2.4, there are three types of resampling algorithms: oversampling, undersampling and the hybrid ones. In the scope of this work, the focus will be the oversampling and hybrid strategies.

Table 25 – Proposed resampling algorithms

Algorithm	Type
ROS	Oversampling
SMOTE	Oversampling
Borderline-SMOTE	Oversampling
ADASYN	Oversampling
SMOTE + ENN	Hybrid (Oversampling + Undersampling)
SMOTE + Tomek Links	Hybrid (Oversampling + Undersampling)

4.5 Hierarchical Classification Approaches

In this work, the focus will be to investigate the hierarchical classification through the use of local approaches. That includes the LCPN and the LCN, the two most common local approaches in the literature.

In addition to this, there are other important definitions to help us to categorize a hierarchical classification algorithm. The work of (SILLA; FREITAS, 2011) proposes a unifying framework to categorize hierarchical classification algorithms according to its features, as stated in section section 2.2. The four features that help us categorize the algorithm are: whether it can predict labels in multiple or single paths in the taxonomy, the prediction depth, which hierarchical structure the algorithm can handle.

According to the four features, we may define the algorithms being implemented in this work as:

- SPP: That means we are looking to predict classes that have only one possible path in the hierarchy.
- MLNP: we are looking for an algorithm that is capable of predicting full depth labels. This means that only leaf nodes will be investigated.
- Tree: the datasets used in this work are suitable to be arranged in a tree structure.
- LCPN and LCN: in this work, we propose to use both LCPN and LCN approaches.

4.5.1 Local Classifier per Parent Node (LCPN)

The LCPN hierarchical classification approach is one of the local ones to be evaluated in this work. The main differences between the different approaches are: how the data is retrieved and relabeled in the taxonomy, which was explained earlier in sections 4.3.1.1 and 4.3.1.2, and the training and prediction phases of the hierarchical classification task. The current section aims to explain these two phases in more detail for the LCPN scenario.

4.5.1.1 Training

The training process of the LCPN works similarly to a depth-first traversal. Starting at the root (R) node, the training function trains the node marked as parent and goes down the hierarchy by visiting each of its child nodes until it reaches a leaf node. When it reaches a leaf node, the process returns to the previous parent node to verify if there are any other nodes need to be trained, until it reaches the last leaf node of the tree. Figure 23 adapts the hierarchy showed in figure 18 to show which parent classifiers are trained and in which order. The parent nodes are highlighted with a different border width. In this case, R/Fungal is a parent node with only one child, thus there is no need for a parent node classifier in this node.

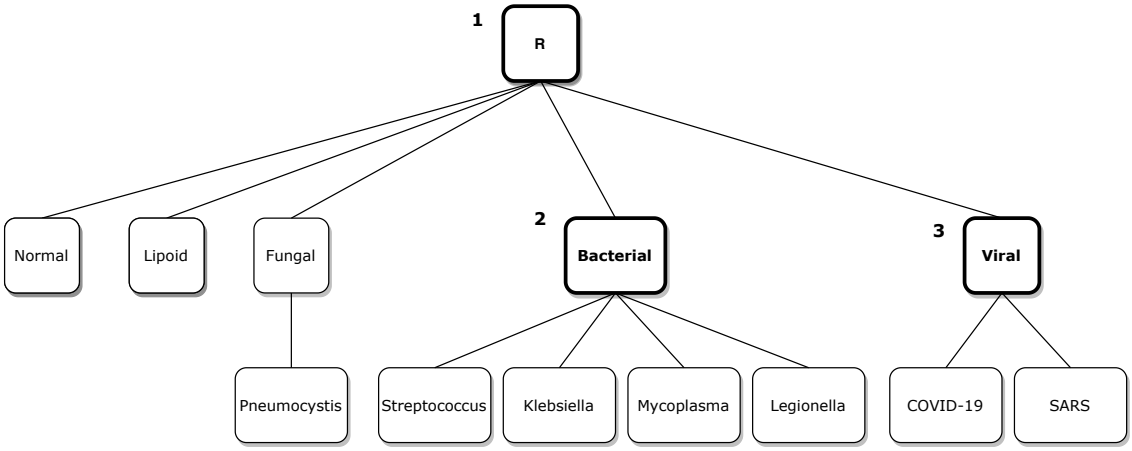


Figure 23 – Adaptation of the original hierarchy to show the trained nodes in the LCPN scenario

4.5.1.2 Prediction

The prediction process for the LCPN scenario is implemented using a top-down approach, where a prediction is entirely dependent on the one from the previous level. The main reason for doing this is to avoid inconsistencies in the predictions, like predicting COVID-19 as a Bacterial type of pneumonia. In figure 24 we exemplify the prediction for a sample from class COVID-19. In the first parent node classifier (step 1, root of the tree), The prediction indicates that the sample belongs to the R/Viral class. In step 2, the classifier predicts that sample is a COVID-19 sample.

It is important to remember that the example illustrated in 24 is for a leaf node sample. In the case we have a NMLNP problem, such as for an instance from class R/Viral, the prediction could have stopped earlier, in step 2.

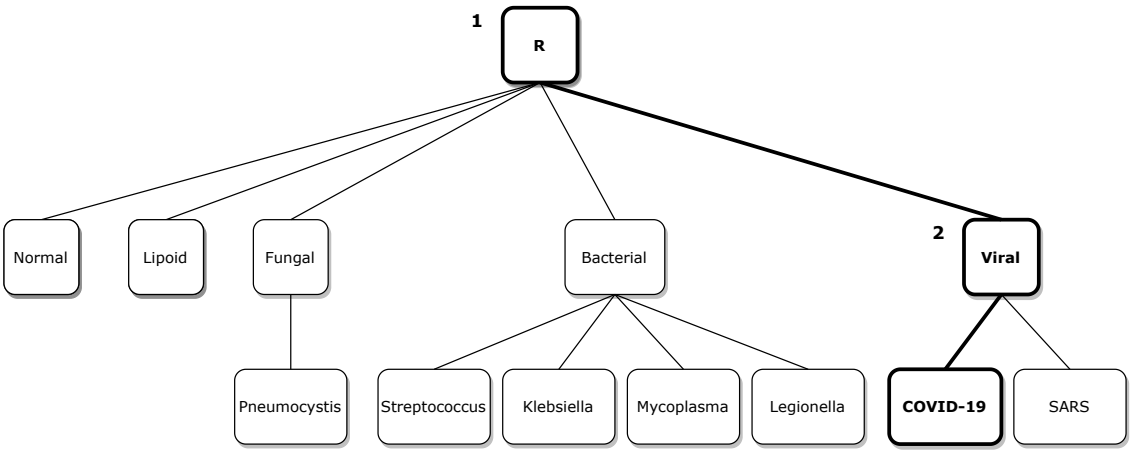


Figure 24 – Adaptation of the original hierarchy to show the prediction path in the LCPN scenario

4.5.2 Local Classifier per Node (LCN)

As stated earlier, the local approaches differ between them in the way the data is retrieved, which was explained in section 4.3.1.2, and in the training and prediction process. The current section explains in details how these two phases are performed for the LCN scenario.

4.5.2.1 Training

The training process of the LCN scenario works slightly different from the LCPN one. It starts at the root (R) node by training all of its child nodes. Each one of them will have a binary classifier to identify the sample as belonging to its target class or not. Figure 25 shows the order that the nodes are trained. If one of these nodes has more than one child node, then the training process will go down to the next level of the hierarchy. In this example, the nodes R/Normal, R/Lipoid, R/Fungal didn't have multiple child nodes. However, the R/Bacterial node has it, so the training will continue in the next level of the hierarchy, until it reaches leaf nodes or those with a single child.

4.5.2.2 Prediction

For the LCN scenario, the prediction phase works as it follows: it also starts at root (R) level by checking all the binary classifiers from its child nodes. They are responsible for predicting if a sample belongs or not to the given class. After the decision is made, in the ideal scenario, we will have at least one of the classifiers predicting the positive class. In the example presented in figure 26, this first step is marked as the number 1, where the predicted class was R/Viral. After the first prediction, the prediction phase will continue in the predicted node by repeating the same process as before, by testing its child node

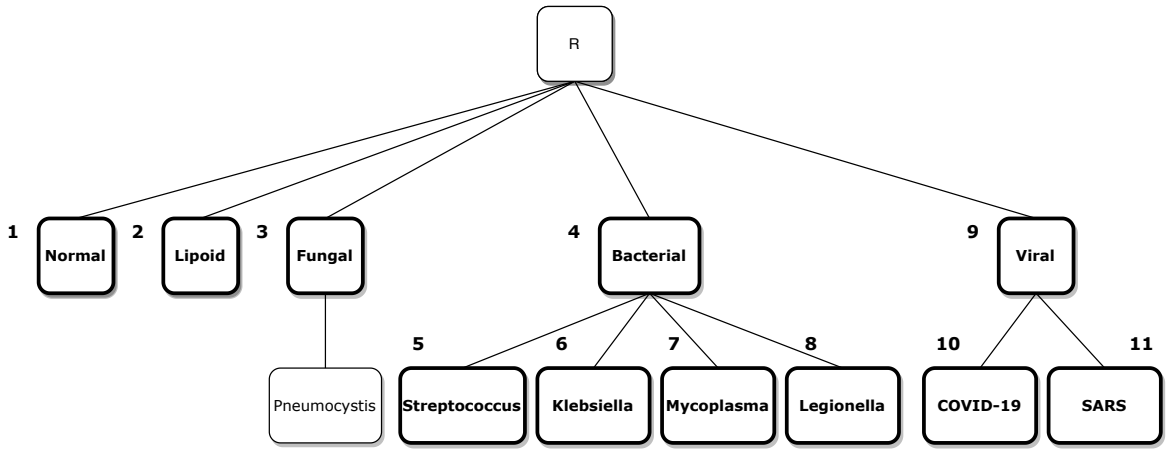


Figure 25 – Adaptation of the original hierarchy to show the trained nodes in the LCN scenario

classifiers. In figure 26, the final predicted class was COVID-19, in the step marked as number 2.

It is important to remember that the predicted path in figure 26 depicts a MLNP problem. If we were to consider a NMLNP problem, the prediction could have stopped in step 1.

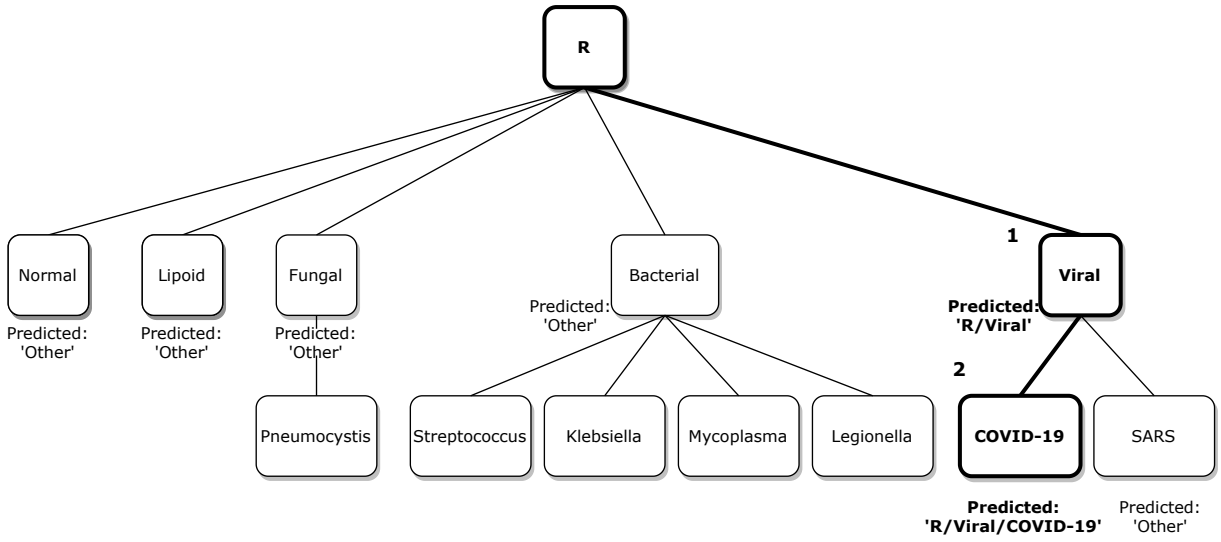


Figure 26 – Adaptation of the original hierarchy to show the prediction path in the LCN scenario

The previous example shows us the flow where no inconsistencies arise. However, in complex data, it may not always be the case. What if we had no positive classes predicted? Or, what if we had more than one binary classifier predicting a positive class? This is an issue called the "blocking problem", and it was briefly described in section 2.3.4. In this case, We would need to consider the probability given by the estimator for each class in

order to resolve these inconsistencies.

One of the cases where an inconsistency may happen during the prediction is when more than one classifier predicts that the sample belongs to the positive class, like the example presented in figure 27. In this case, we have an inconsistent prediction at the COVID-19 and the SARS classifiers. The first one is predicting the instance of data to be a COVID-19 one with a probability of 0.75 and the second predicted that the instance belongs to the SARS class, with a probability of 0.65. This conflict is solved by using the prediction with the highest probability between them, which in this case is the COVID-19 class.

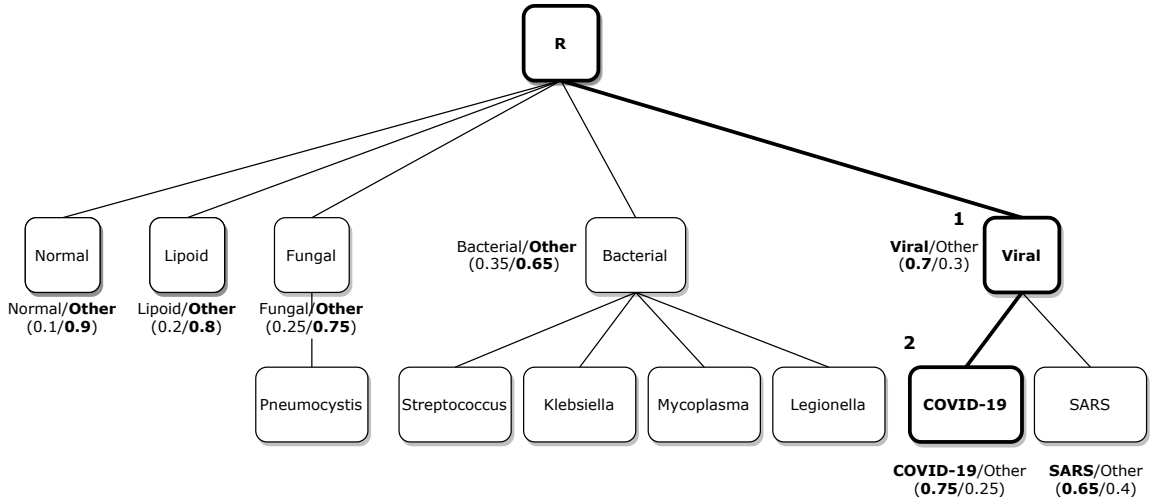


Figure 27 – LCN Path prediction showing a conflict between COVID-19 and SARS classifiers

The second inconsistency scenario that may arise is when none of the classifiers predicts a positive class, which means that none of them is sure if that instance of data belongs to a particular positive class. This scenario is presented in figure 28, and in this case, the COVID-19 and SARS classifiers both classified the sample as 'Other' (the negative class). This problem will be solved in the same way it was solved for the scenario where both classifiers predicted the positive class: by using the highest probability returned by the estimators. In this example from figure 28, COVID-19 is predicted with a 0.45 probability while SARS had a 0.3. It is more likely that the sample is a COVID-19 one, because of this, the highest probability will be used.

4.6 Evaluation Procedure

The evaluation procedure to assess the effectiveness of the use of different resampling approaches and algorithms in the hierarchical classification task will make use of some of the most common metrics to evaluate classification problems. For the current work

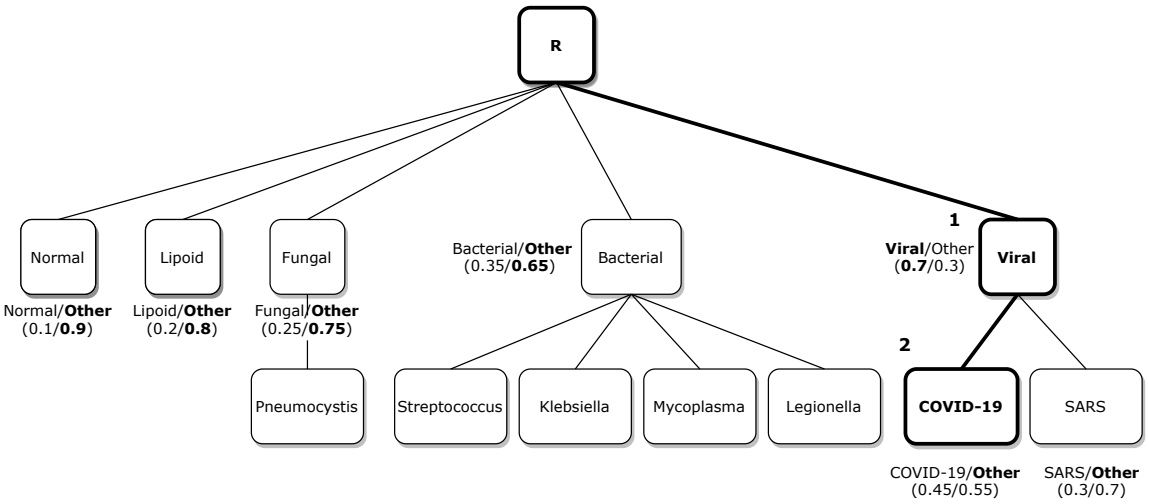


Figure 28 – LCN prediction path showing the classifiers were not able to classify in COVID-19 or SARS

in particular, which employs highly imbalanced datasets, using Accuracy wouldn't be adequate. If hypotetically, the RYDLS C-19 IDC dataset had 95% of the samples belonging to the R/Normal class, and a trivial classifier predicted all of the samples to be normal, we would still have 95% accuracy, but the interest classes represented by the lung infections wouldn't have any correct predictions. To avoid this kind of issue, different classical metrics are employed. They are the following ones:

- Precision: It may be defined as the number of true positives (T_p) divided by the sum of true positives (T_p) and false positives (F_p).In other words, it aims to calculate the proportion of positive identifications that were classified correctly.

$$P = \frac{T_p}{T_p + F_p} \tag{4.3}$$

- Recall: It may be defined as the number of true positives (T_p) divided by the sum of true positives (T_p) and false negatives (F_n) In other words, it aims to calculate the proportion of the actual positive identifications were classified correctly.

$$R = \frac{T_p}{T_p + F_n} \tag{4.4}$$

- F1-Score: It is calculated as the harmonic mean between the precision and recall:

$$F1 = \frac{2 * P * R}{P + R} \tag{4.5}$$

These are the versions suitable for binary classifiers, for multi-class classifiers we have the micro and macro-averaged versions. The micro-averaged (VATEEKUL; KUBAT; SARINNAPAKORN, 2014) version may be defined as it follows:

$$P_\mu = \frac{\sum_{c_i \in C} T_p(c_i)}{\sum_{c_i \in C}(T_p(c_i) + F_p(c_i))} \tag{4.6}$$

$$R_\mu = \frac{\sum_{c_i \in C} T_p(c_i)}{\sum_{c_i \in C}(T_p(c_i) + F_n(c_i))} \tag{4.7}$$

$$F1_\mu = \frac{2 * P_\mu * R_\mu}{P_\mu * + R_\mu} \tag{4.8}$$

Where c_i is the i^{th} class of the set of classes C, which represent all the classes available in the dataset, $T_p(c_i)$ the number of true positives for the i^{th} class, $F_p(c_i)$ the number of false positives for the same class and $F_n(c_i)$ the number of false negatives. However, there is a problem with the micro-averaged version. The issue is that it is a lot more difficult to notice changes on its values when the the minority classes are resampled, because after resampling the data majority classes often suffer a decrease in performance, which compromises the final score. This happens because the classes doesn't have the same weight in the final score. Because of this, we propose to use the macro-averaged (VATEEKUL; KUBAT; SARINNAPAKORN, 2014) version of the metrics:

$$P_M = \frac{\sum_{c_i \in C} P(c_i)}{n_C} \tag{4.9}$$

$$R_M = \frac{\sum_{c_i \in C} R(c_i)}{n_C} \tag{4.10}$$

$$F1_M = \frac{2 * P_M * R_M}{P_M + R_M} \tag{4.11}$$

Where n_C represents the total number of classes from the set C, $P(c_i)$ the precision for ith class of the set C and $R(c_i)$ the recall for the same class. The macro-averaged version of the metrics basically computes the metrics for each class and then it calculates an average value based on the number of different classes. In this way, each one of the classes has the same weight and thus the impact of resampling will be measured equally throughout the classes.

Evaluating hierarchical classification algorithms is still an open question (FENG; FU; ZHENG, 2017). Even though there are authors that use these classic metrics to evaluate them, sometimes they are not the most suited for this kind of problem (KIRITCHENKO et al., 2006), once they do not take into account hierarchy of the label. However, the

hierarchical metrics proposed by (KIRITCHENKO et al., 2006) and explained in detail in section 2.3.5 do not take into account the class imbalance and thus they might not be the most appropriate for the scenario being explored in the current work. Because of this, we propose to use the macro-averaged F1 score as the main metric used in the evaluation of the models.

Another part of the evaluation procedure is the how the different algorithms and approaches will be compared among them. As part of this work, we propose to compare:

- Compare the different resampling approaches (Flat, Local, Threshold Selective and Local Selective, or the baseline) between them, for both LCPN and LCN scenarios.
- Compare the use of different resampling algorithms for both LCPN and LCN when used together with the aforementioned resampling approaches. The list of algorithms to be used is the following: none (baseline), SMOTE, Borderline-SMOTE, SMOTE-ENN, SMOTE-Tomek, ADASYN and ROS.

The most reliable way to compare them is through the use of statistical non-parametric tests. The Wilcoxon Signed-Rank test will be used to individually test if each one of the proposed resampling aproaches yields statistically significant improvement (WILCOXON, 1945). We also proposed to use Friedman's non-parametric test will be used to check if there is significant difference between the algorithms or any of the different approaches. If the difference is confirmed, use Nemenyi as a post-hoc test to check which algorithms or approaches are the best ones (DEMŠAR, 2006).

5 Results and Discussion

In this chapter, we present and analyze our results. To obtain them, we employed the methodology presented in chapter 4 to the datasets from section 4.1. In these experiments, our objective was to assess the use of different resampling approaches and different resampling algorithms and compare it with the baseline, where no resampling techniques were used. In these experiments, the datasets from section 4.1 were used.

The experimental configurations are summarized in table 26. In all experiments, we used the following classification algorithms: Decision Trees (DT), Random Forest (RF), Multilayer Perceptron (MLP), Support Vector Machine (SVM), Naive Bayes (NB) and K-Nearest Neighbors (kNN). Another important fact is that we used the Stratified Cross-Validation technique (with k = 5), where each Cross-Validation split could be considered a different experiment. With this information in mind, it is possible to say that a total of 1020 experiments for each of the six datasets were executed, which means a total of 6120 experiments.

It is noteworthy that some of the preliminary results obtained during the development of this work were used in a paper that was accepted and presented at the 21st IEEE International Conference on BioInformatics and BioEngineering (BARROS et al., 2021).

Table 26 – Summary of the experiments performed

Hierarchical Classifier	**Resampling**	**Resampling Algorithms**	**Classification Algorithm**	**No. of Experiments (per dataset)**
LCPN	None	None	[DT, RF, MLP, SVM, NB, KNN]	30
	Flat	[ROS, SMOTE, Borderline-SMOTE, ADASYN, SMOTE-ENN, SMOTE-Tomek]	[DT, RF, MLP, SVM, NB, KNN]	180
	Local	[ROS, SMOTE, Borderline-SMOTE, ADASYN, SMOTE-ENN, SMOTE-Tomek]	[DT, RF, MLP, SVM, NB, KNN]	180
	Local Selective	[ROS, SMOTE, Borderline-SMOTE, ADASYN, SMOTE-ENN, SMOTE-Tomek]	[DT, RF, MLP, SVM, NB, KNN]	30
LCN	None	None	[DT, RF, MLP, SVM, NB, KNN]	30
	Flat	[ROS, SMOTE, Borderline-SMOTE, ADASYN, SMOTE-ENN, SMOTE-Tomek]	[DT, RF, MLP, SVM, NB, KNN]	180
	Local	[ROS, SMOTE, Borderline-SMOTE, ADASYN, SMOTE-ENN, SMOTE-Tomek]	[DT, RF, MLP, SVM, NB, KNN]	180
	Threshold Selective	[ROS, SMOTE, Borderline-SMOTE, ADASYN, SMOTE-ENN, SMOTE-Tomek]	[DT, RF, MLP, SVM, NB, KNN]	180
	Local Selective	[ROS, SMOTE, Borderline-SMOTE, ADASYN, SMOTE-ENN, SMOTE-Tomek]	[DT, RF, MLP, SVM, NB, KNN]	30

In order to make it easier to understand how the analysis is going to be presented, we propose the definition of a new terminology that will simplify how the results are

presented. We propose to concatenate the resampling approach column with the resampling algorithms column from table 26 and call the result of it as resampling strategy. So, for example, if we used the flat resampling with SMOTE algorithm, we used a Flat-SMOTE resampling strategy. In the case of the Local Selective resampling, we will refer the strategy as Local Selective only, because the resampling algorithm used is dynamic as already mentioned in section 4.4.4, and it depends on the experiment execution. Besides this, we also would like to make it clear that each of these resampling strategies was used together with each one of the classifiers from the classification algorithm column. The summary of this new terminology is presented in table 27.

Table 27 – Summary of the new terminology used in the result analysis

Hierarchical Classifier	**Resampling Strategy**	**Classification Algorithm**
LCPN	Baseline	[DT, RF, MLP, SVM, NB, KNN]
	[Flat-ROS, Flat-SMOTE, Flat-Borderline, Flat-ADASYN, Flat-SMOTE-ENN, Flat-SMOTE-Tomek]	[DT, RF, MLP, SVM, NB, KNN]
	[Local-ROS, Local-SMOTE, Local-Borderline, Local-ADASYN, Local-SMOTE-ENN, Local-SMOTE-Tomek]	[DT, RF, MLP, SVM, NB, KNN]
	Local Selective	[DT, RF, MLP, SVM, NB, KNN]
LCN	Baseline	[DT, RF, MLP, SVM, NB, KNN]
	[Flat-ROS, Flat-SMOTE, Flat-Borderline, Flat-ADASYN, Flat-SMOTE-ENN, Flat-SMOTE-Tomek]	[DT, RF, MLP, SVM, NB, KNN]
	[Local-ROS, Local-SMOTE, Local-Borderline, Local-ADASYN, Local-SMOTE-ENN, Local-SMOTE-Tomek]	[DT, RF, MLP, SVM, NB, KNN]
	[Threshold-ROS, Threshold-SMOTE, Threshold-Borderline, Threshold-ADASYN, Threshold-SMOTE-ENN, Threshold-SMOTE-Tomek]	[DT, RF, MLP, SVM, NB, KNN]
	Local Selective	[DT, RF, MLP, SVM, NB, KNN]

5.1 Experimental Setup

Following the methodology defined in section 4, we applied the stratified k-fold cross-validation technique, randomly splitting our dataset in $k = 5$ stratified folds, where each fold maintains the proportion of number of samples per class from the original dataset (KOHAVI, 1995). For each iteration of the k-fold procedure, one of these folds is selected for evaluating the classifier while the other four are used for training the classifier.

For all of the resampling algorithms that are based on a Nearest Neighbors algorithm (e.g. SMOTE), the number of neighbors used was $k = 5$.

As mentioned in Section 4.1, we filter our dataset to exclude any classes with less than 7 samples in that class. Considering some resampling algorithms that use a Nearest Neighbors strategy to oversample, and we define our $k = 5$, there is a minimum

requirement of $k + 1 = 6$ samples per class in the conjuction of all training folds. But as we are using a stratified k-fold algorithm, there will be at least one sample per class in the test fold. Thus we need at least $k + 2 = 7$ samples per class in our dataset.

The classification algorithms used for these experiments as well as the hyper-parameter configurations used in them are listed in table 54 of Appendix B. In all of them the scikit-learn's[1] implementation was used.

For the non-parametric statistical tests that were performed during the analysis, we utilize the SciPy-Stats Module[2]. To have a better visualization of the Nemenyi post-hoc tests' results, we utilize the Orange Data Mining Library[3].

Lastly, we focused our experiments on hierarchical classification problems structured in trees, with mandatory leaf node prediction (MLNP) and we used the siblings policy for the LCPN experiments and the less-inclusive policy for the LCN experiments.

5.2 Classification Results

First, we present in figure 29 the mean macro-averaged F1 score across all datasets and all classifiers, for both LCPN and LCN experiments. These values were calculated based on the scores obtained from the experiments with different classifiers and different resampling approaches. We present these values in the "Avg." column in tables 55 and 62 in Appendix B.

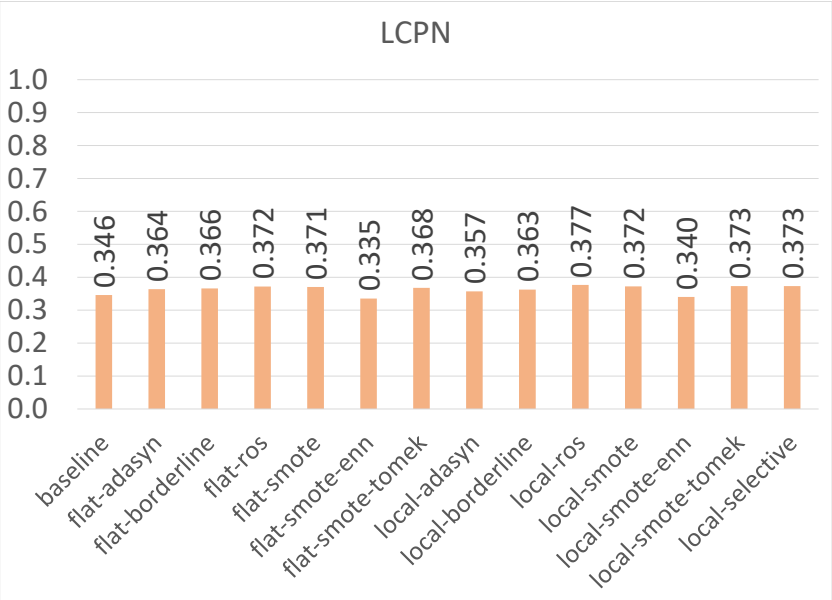
Besides calculating the mean values for all datasets, we also calculated the mean values for each dataset separately. In figures 30 and 31, the mean macro-averaged F1 Score for the LCPN and LCN classifiers is presented, where each subfigure represents a the results for a single dataset. The results for each dataset used to calculate these mean values are presented in tables 56 through 61 for the LCPN experiments and 63 through 68 for the LCN experiments. All of these tables are available in appendix B.

5.3 Discussion and Analysis

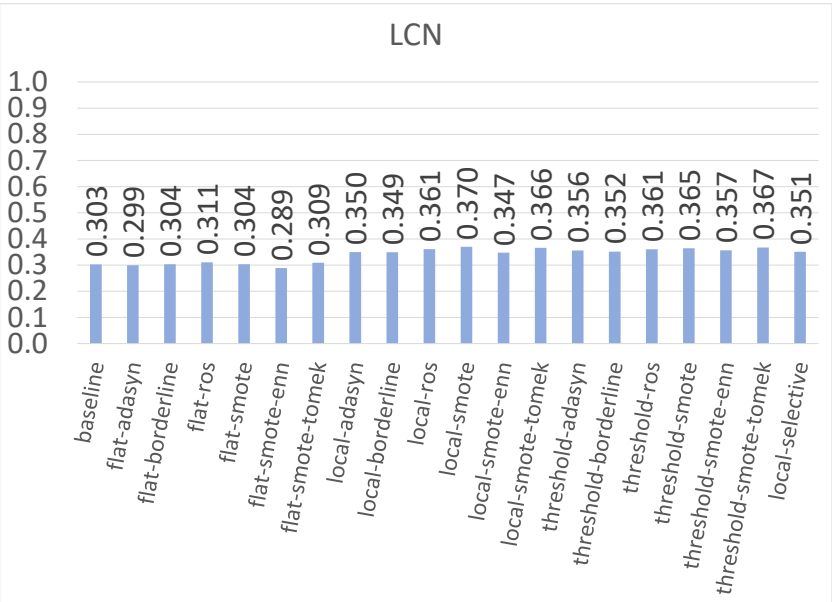
In order to analyze and discuss the results, let us recapitulate that the hypothesis of this work (presented in section 1.2) is:

- ***Hypothesis***: The proposed resampling approaches outpeform the baseline approach (without resampling) for hierarchical classification problems.

[1] <https://scikit-learn.org>
[2] <https://docs.scipy.org/doc/scipy/reference/stats.html>
[3] <https://orange-data-mining-library.readthedocs.io/en/latest/reference/evaluation.cd.html>

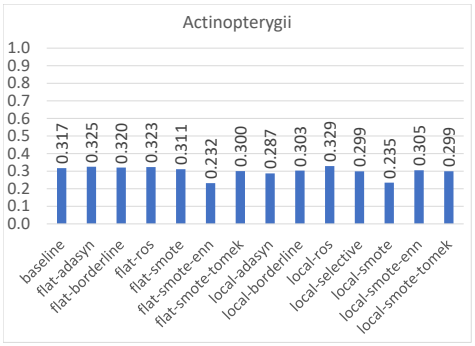


(a) LCPN

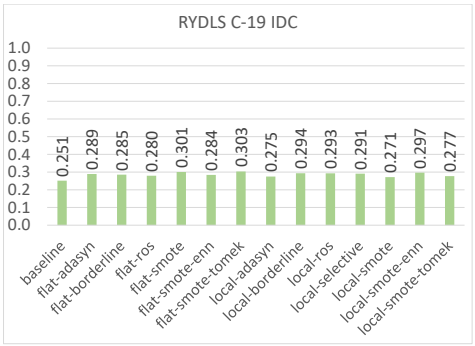


(b) LCN

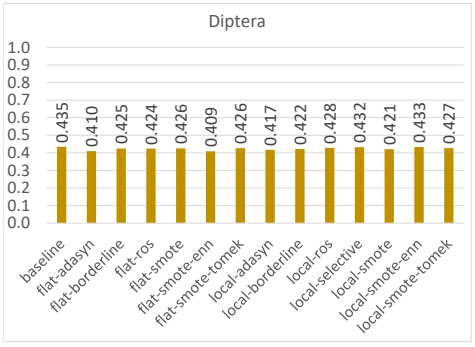
Figure 29 – Mean macro-averaged F1 score across all datasets



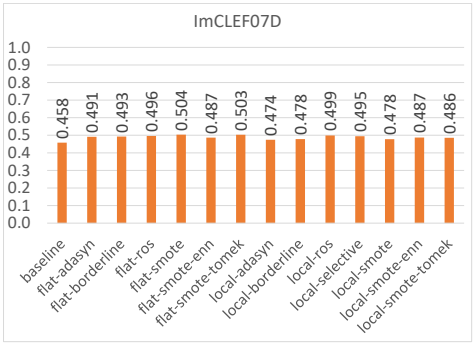
(a) Actinopterygii



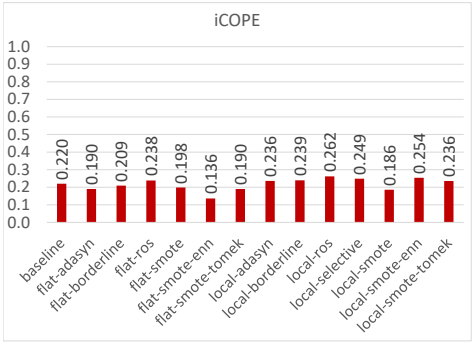
(b) RYDLS C-19 IDC



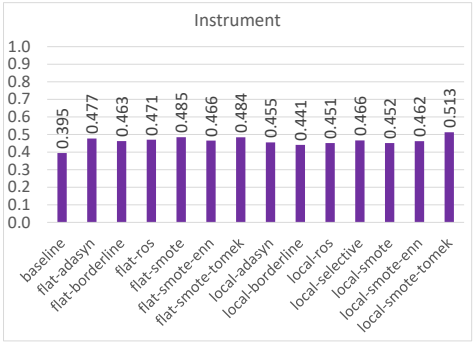
(c) Diptera



(d) ImCLEF07D



(e) iCOPE

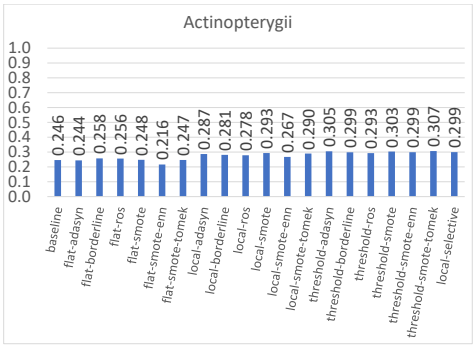


(f) Instrument

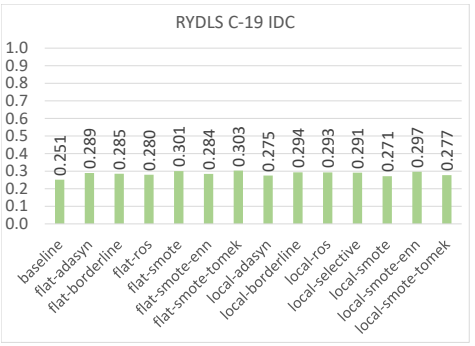
Figure 30 – Mean macro-averaged F1 score results for the LCPN experiments in each dataset

Which we can split in two statements: a Null hypothesis H_0 and an alternative hypothesis H_1:

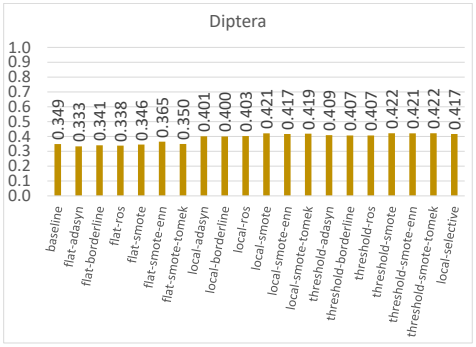
- H_0: The proposed resampling approaches do not outpeform the baseline approach (without resampling) for hierarchical classification problems.
- H_1: The proposed resampling approaches outpeform the baseline approach (without



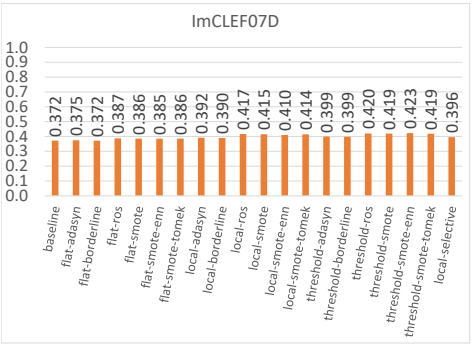
(a) Actinopterygii



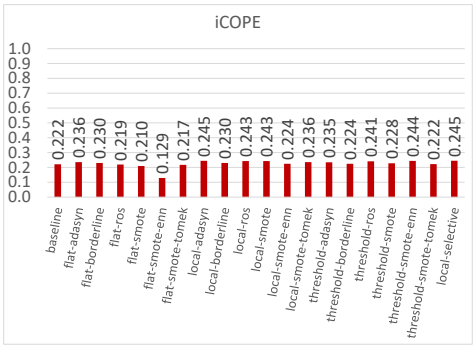
(b) RYDLS C-19 IDC



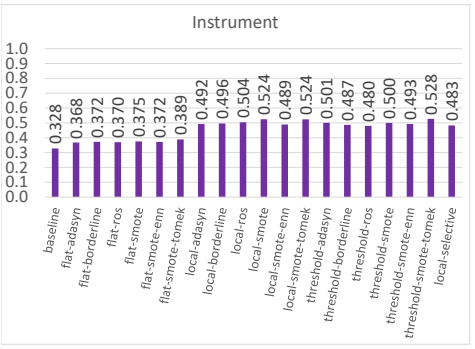
(c) Diptera



(d) ImCLEF07D



(e) iCOPE



(f) Instrument

Figure 31 – Mean macro-averaged F1 score results for the LCN experiments in each dataset

resampling) for hierarchical classification problems.

In order to prove that the alternative hypothesis is valid, we consider the results obtained in the classification experiments, presented in section 5.2, and we use non-parametric statistical tests to either prove or refute the hypothesis. In all the statistical tests in this analysis we used a significance of 95 % ($\alpha = 0.05$).

First, we need to prove that each of the resampling strategies employed in the

experiments was effective and improved the classification results compared to the baseline. To test this, we propose to use a Wilcoxon Signed Rank test for paired samples (WILCOXON, 1945). In this test we filter the baseline results and compare to each of the resampling strategies individually. In this case, our samples are the macro-averaged F1 score results obtained for the baseline and for each resampling strategy. We use this test because we ran experiments with the same classification algorithms, with exactly the same data folds and also the same datasets for all the resampling approaches.

Besides testing if there was statistically significant difference between the experiments with resampling and the baseline, in order to accomodate those cases where we could not observe statistically significant improvement over the baseline, we also calculate the difference between each strategy and the baseline. This calculation tells us if there was any improvement over the baseline, even in those cases that are not statistically significant.

Lastly, to check which resampling strategy was the best ranked, we also calculated the average ranks of each strategy using the Friedman non-parametric test together with Nemenyi as a post-hoc (DEMŠAR, 2006). With this test we were able to check which of the approaches presented the best performance for the task at hand. A summary of all the tests applied to the results and how they were performed is presented in table 28. It is important to remember that each one of these tests was executed for all datasets at once and also for each dataset separately. We also analyzed the results for the LCPN and LCN experiment separately.

Table 28 – Tests performed during the analysis

Test	Procedure	Objective
Wilcoxon Signed-Rank	Compute a pair-wise comparison between the baseline and each of the resampling strategies.	Check if there is statistically significant difference between the baseline and each of the resampling strategies
Baseline difference	Compute the difference between each resampling strategy and the baseline.	Check if there was improvement over the baseline
Friedman/Nemenyi	Compute a comparison between multiple groups, where each group is a different resampling strategy.	Check if there is statistically significant difference between the resampling approaches, including the baseline and also compute the average ranks and obtain a CD plot.

5.3.1 Wilcoxon Test

The objective of the first test was checking if there was statistically significant difference between the baseline and each one of the resampling strategies employed in the experiments, which means that we fixed the baseline results and tested against each of the strategies. We used the Wilcoxon Signed-Rank test with the alternative hypothesis being that the results where resampling was used were greater than the baseline (WILCOXON, 1945). The results of the test are the p-values. The results for the LCPN experiments are

presented in tables 29 (all datasets) and 30 (for each dataset). The highlighted values indicate p-values less than α ($\alpha = 0.05$), once we are using a 95% significance.

Table 29 – Results for the Wilcoxon Signed-Rank test applied to the LCPN experiments. Highlighed values mean $p < \alpha$ ($\alpha = 0.05$)

Strategy	p-value
flat-adasyn	1.43×10^{-1}
flat-borderline	8.70×10^{-4}
flat-ros	2.20×10^{-5}
flat-smote	1.47×10^{-3}
flat-smote-enn	9.73×10^{-1}
flat-smote-tomek	1.93×10^{-2}
local-adasyn	2.01×10^{-1}
local-borderline	1.41×10^{-2}
local-ros	8.96×10^{-8}
local-selective	3.65×10^{-3}
local-smote	7.49×10^{-5}
local-smote-enn	7.98×10^{-1}
local-smote-tomek	2.23×10^{-5}

Table 30 – Results for the Wilcoxon Signed-Rank test applied to the LCPN experiments, for each dataset. Highlighed values mean $p < \alpha$ ($\alpha = 0.05$)

Strategy	Actinopterygii	RYDLS C-19 IDC	Diptera	ImCLEF07D	iCOPE	Instrument
flat-adasyn	8.20×10^{-1}	1.66×10^{-2}	1.00	3.44×10^{-2}	9.91×10^{-1}	1.11×10^{-4}
flat-borderline	2.16×10^{-2}	3.44×10^{-2}	1.00	1.80×10^{-3}	7.41×10^{-1}	2.10×10^{-4}
flat-ros	8.38×10^{-4}	7.94×10^{-2}	1.00	3.42×10^{-3}	1.91×10^{-1}	3.76×10^{-5}
flat-smote	6.87×10^{-1}	2.34×10^{-3}	1.00	1.38×10^{-3}	9.37×10^{-1}	3.97×10^{-5}
flat-smote-enn	1.00	2.36×10^{-2}	1.00	5.55×10^{-2}	1.00	6.43×10^{-4}
flat-smote-tomek	9.58×10^{-1}	4.81×10^{-4}	9.98×10^{-1}	4.64×10^{-3}	9.95×10^{-1}	4.09×10^{-5}
local-adasyn	1.00	9.23×10^{-2}	1.00	8.90×10^{-2}	3.13×10^{-1}	2.64×10^{-4}
local-borderline	9.84×10^{-1}	2.66×10^{-3}	1.00	6.02×10^{-2}	2.33×10^{-1}	2.06×10^{-3}
local-ros	6.80×10^{-2}	4.11×10^{-3}	9.80×10^{-1}	1.31×10^{-4}	1.50×10^{-2}	1.29×10^{-3}
local-selective	9.82×10^{-1}	1.75×10^{-2}	9.88×10^{-1}	6.60×10^{-3}	2.20×10^{-1}	1.31×10^{-4}
local-smote	9.63×10^{-1}	2.66×10^{-3}	8.36×10^{-1}	3.21×10^{-3}	5.55×10^{-2}	4.09×10^{-5}
local-smote-enn	9.99×10^{-1}	5.78×10^{-2}	9.97×10^{-1}	8.25×10^{-2}	9.62×10^{-1}	4.81×10^{-4}
local-smote-tomek	8.89×10^{-1}	1.80×10^{-3}	7.86×10^{-1}	1.35×10^{-2}	1.22×10^{-2}	1.53×10^{-4}

For the LCPN experiments, we can see in table 29 that considering all datasets, the following resampling strategies improved the classification: Flat-Borderline, Flat-ROS, Flat-SMOTE, Flat-SMOTE-Tomek, Local-Borderline, Local-ROS, Local-Selective, Local-SMOTE and Local-SMOTE-Tomek. However, if we look at table 30, we can see that this improvement is valid only in particular datasets. In the Diptera dataset, resampling did not increase the classification scores with statistical significance for any scenarios. For the Actinopterygii, RYDLS C19-IDC, ImCLEF07D and iCOPE datasets, there was improvement only in a few cases. The only dataset where there was improvement with every strategy was in the Instrument dataset. In total, there are 78 scenarios (13 strategies for 6 datasets), and the results reported statistically significant improvement in 36 out of the 78 scenarios (46.15%).

For the LCN experiments, we had in total 114 scenarios (19 strategies for 6 datasets), the results of the analysis are presented in tables 32 (all datasets) and 33 (each dataset).

Table 31 – Results for the Wilcoxon Signed-Rank test applied to the LCN experiments. Highlighed values mean $p < \alpha$ ($\alpha = 0.05$)

Strategy	p-value
flat-adasyn	8.40×10^{-1}
flat-borderline	3.01×10^{-1}
flat-ros	1.61×10^{-1}
flat-smote	3.40×10^{-1}
flat-smote-enn	8.78×10^{-1}
flat-smote-tomek	8.08×10^{-2}
local-adasyn	7.71×10^{-9}
local-borderline	5.07×10^{-9}
local-ros	2.09×10^{-13}
local-selective	5.68×10^{-12}
local-smote	2.69×10^{-17}
local-smote-enn	4.17×10^{-10}
local-smote-tomek	5.05×10^{-14}
threshold-adasyn	3.62×10^{-10}
threshold-borderline	2.90×10^{-10}
threshold-ros	2.12×10^{-14}
threshold-smote	5.40×10^{-15}
threshold-smote-enn	1.26×10^{-12}
threshold-smote-tomek	5.34×10^{-15}

Table 32 – Results for the Wilcoxon Signed-Rank test applied to the LCN experiments. Highlighed values mean $p < \alpha$ ($\alpha = 0.05$)

Strategy	p-value (LCPN)	p-value (LCN)
flat-adasyn	1.43×10^{-1}	8.40×10^{-1}
flat-borderline	8.70×10^{-4}	3.01×10^{-1}
flat-ros	2.20×10^{-5}	1.61×10^{-1}
flat-smote	1.47×10^{-3}	3.40×10^{-1}
flat-smote-enn	9.73×10^{-1}	8.78×10^{-1}
flat-smote-tomek	1.93×10^{-2}	8.08×10^{-2}
local-adasyn	2.01×10^{-1}	7.71×10^{-9}
local-borderline	1.41×10^{-2}	5.07×10^{-9}
local-ros	8.96×10^{-8}	2.09×10^{-13}
local-selective	3.65×10^{-3}	5.68×10^{-12}
local-smote	7.49×10^{-5}	2.69×10^{-17}
local-smote-enn	7.98×10^{-1}	4.17×10^{-10}
local-smote-tomek	2.23×10^{-5}	5.05×10^{-14}
threshold-adasyn	n/a	3.62×10^{-10}
threshold-borderline	n/a	2.90×10^{-10}
threshold-ros	n/a	2.12×10^{-14}
threshold-smote	n/a	5.40×10^{-15}
threshold-smote-enn	n/a	1.26×10^{-12}
threshold-smote-tomek	n/a	5.34×10^{-15}

In table 32, we notice that considering all the datasets, there was no improvement in those experiments that used the Flat Resampling strategy, while in all the other ones there was statistically significant improvement. However, if we look at table 33, we can see

Table 33 – Results for the Wilcoxon Signed-Rank test applied to the LCN scenario. Highlighed values mean $p < \alpha$ ($\alpha = 0.05$)

Strategy	Actinopterygii	RYDLS C-19 IDC	Diptera	ImCLEF07D	iCOPE	Instrument
flat-adasyn	7.35×10^{-1}	9.91×10^{-1}	1.00	2.72×10^{-1}	3.59×10^{-1}	9.80×10^{-3}
flat-borderline	4.00×10^{-4}	9.94×10^{-1}	9.91×10^{-1}	4.39×10^{-1}	4.07×10^{-1}	3.40×10^{-3}
flat-ros	4.49×10^{-2}	8.20×10^{-1}	1.00	5.78×10^{-2}	5.86×10^{-1}	3.00×10^{-3}
flat-smote	1.50×10^{-1}	9.83×10^{-1}	9.84×10^{-1}	6.80×10^{-2}	7.55×10^{-1}	7.80×10^{-3}
flat-smote-enn	9.96×10^{-1}	9.80×10^{-1}	7.00×10^{-4}	4.49×10^{-2}	1.00	3.90×10^{-3}
flat-smote-tomek	3.91×10^{-1}	9.66×10^{-1}	3.00×10^{-1}	7.64×10^{-2}	5.78×10^{-1}	4.00×10^{-4}
local-adasyn	1.31×10^{-4}	6.17×10^{-1}	6.91×10^{-4}	5.55×10^{-2}	1.64×10^{-1}	6.19×10^{-6}
local-borderline	3.58×10^{-4}	5.37×10^{-1}	4.47×10^{-4}	9.58×10^{-2}	2.33×10^{-1}	1.18×10^{-6}
local-ros	1.29×10^{-3}	1.91×10^{-1}	4.86×10^{-5}	2.10×10^{-4}	5.78×10^{-2}	2.36×10^{-6}
local-selective	1.19×10^{-5}	7.00×10^{-1}	1.30×10^{-6}	6.99×10^{-3}	7.07×10^{-2}	9.60×10^{-7}
local-smote	6.19×10^{-6}	1.11×10^{-1}	8.67×10^{-7}	1.11×10^{-4}	9.23×10^{-2}	8.67×10^{-7}
local-smote-enn	5.18×10^{-4}	8.31×10^{-1}	8.67×10^{-7}	1.79×10^{-4}	6.41×10^{-1}	8.67×10^{-7}
local-smote-tomek	2.90×10^{-5}	3.52×10^{-1}	8.67×10^{-7}	1.03×10^{-4}	2.45×10^{-1}	8.67×10^{-7}
threshold-adasyn	8.99×10^{-6}	6.71×10^{-1}	3.08×10^{-4}	1.66×10^{-2}	1.54×10^{-1}	1.70×10^{-5}
threshold-borderline	2.04×10^{-5}	6.63×10^{-1}	1.42×10^{-4}	1.35×10^{-2}	3.85×10^{-1}	1.76×10^{-6}
threshold-ros	3.86×10^{-4}	1.57×10^{-1}	1.19×10^{-5}	8.02×10^{-5}	8.25×10^{-2}	3.49×10^{-6}
threshold-smote	8.20×10^{-6}	2.69×10^{-1}	8.67×10^{-7}	6.80×10^{-5}	3.83×10^{-1}	3.49×10^{-6}
threshold-smote-enn	8.99×10^{-6}	9.44×10^{-1}	8.67×10^{-7}	2.04×10^{-5}	1.31×10^{-1}	4.23×10^{-6}
threshold-smote-tomek	3.17×10^{-6}	3.77×10^{-1}	8.67×10^{-7}	7.39×10^{-5}	6.17×10^{-1}	8.67×10^{-7}

that the difference was not observed in all datasets. For the RYDLS C-19 IDC and the iCOPE datasets, there was no improvement at all. In all the other datasets, the difference was mostly concentrated around the Local, Local Selective and Threshold approaches, with this last two being approaches proposed in this work. Considering a total of 114 possible scenarios, there was statistically significant improvement in 60 of them (52.63%).

From this first analysis using the Wilcoxon non-parametric test, we may conclude that, overall, resampling the data brings statistically significant improvement to the classification results. It was also possible to observe that even though this is valid in the analysis across all datasets, this is not valid for all of them individually, as depicted in tables 30 and 33.

5.3.2 Difference to the baseline

Besides analyzing if there was statistically significant difference between each of the resampling strategies and the baseline by using the Wilcoxon test, we also analyzed the difference in the macro-averaged F1 scores between the resampling strategies and the baseline. This was done to give us an idea of how many scenarios where at least a minimal improvement was reported, even if it was not statistically significant. First, we analyzed the difference considering all datasets, for both the LCPN and LCN classifiers. To do this, we used the mean scores from tables 55 and 62 and calculated the difference between each result with resampling and the baseline. If the value is positive, it means that resampling improved the scores. If it is negative, it means that scores decreased compared to the baseline. The results are presented in tables 34 and 36.

Table 34 – Difference between the baseline and each resampling strategy for the LCPN experiments, considering all datasets.

Strategy	Difference	(%)
flat-adasyn	0.018	5.20
flat-borderline	0.020	5.78
flat-ros	0.026	7.57
flat-smote	0.025	7.13
flat-smote-enn	-0.011	-3.04
flat-smote-tomek	0.022	6.36
local-adasyn	0.011	3.33
local-borderline	0.017	4.87
local-ros	0.031	8.96
local-selective	0.026	7.61
local-smote	-0.006	-1.59
local-smote-enn	0.027	7.86
local-smote-tomek	0.027	7.86

Table 35 – Difference between the baseline and each resampling strategy for the LCN scenario.

Strategy	Difference	(%)
flat-adasyn	-0.004	-1.16
flat-borderline	0.001	0.28
flat-ros	0.008	2.64
flat-smote	0.001	0.28
flat-smote-enn	-0.014	-4.68
flat-smote-tomek	0.006	2.04
local-adasyn	0.047	15.64
local-borderline	0.046	15.36
local-ros	0.058	19.27
local-smote	0.067	22.25
local-smote-enn	0.045	14.76
local-smote-tomek	0.063	20.87
threshold-adasyn	0.053	17.62
threshold-borderline	0.049	16.19
threshold-ros	0.058	19.11
threshold-smote	0.062	20.43
threshold-smote-enn	0.054	17.90
threshold-smote-tomek	0.065	21.37
local-selective	0.048	16.02

Table 36 – Difference between the baseline and each resampling strategy for the LCN scenario.

Strategy	difference (LCPN)	% (LCPN)	difference (LCN)	% (LCN)
flat-adasyn	0.018	5.20	-0.004	-1.16
flat-borderline	0.020	5.78	0.001	0.28
flat-ros	0.026	7.57	0.008	2.64
flat-smote	0.025	7.13	0.001	0.28
flat-smote-enn	-0.011	-3.04	-0.014	-4.68
flat-smote-tomek	0.022	6.36	0.006	2.04
local-adasyn	0.011	3.33	0.047	15.64
local-borderline	0.017	4.87	0.046	15.36
local-ros	0.031	8.96	0.058	19.27
local-smote	0.026	7.61	0.067	22.25
local-smote-enn	-0.006	-1.59	0.045	14.76
local-smote-tomek	0.027	7.86	0.063	20.87
local-selective	0.027	7.86	0.048	16.02
threshold-adasyn	n/a	n/a	0.053	17.62
threshold-borderline	n/a	n/a	0.049	16.19
threshold-ros	n/a	n/a	0.058	19.11
threshold-smote	n/a	n/a	0.062	20.43
threshold-smote-enn	n/a	n/a	0.054	17.90
threshold-smote-tomek	n/a	n/a	0.065	21.37

In the differences reported in tables 34 and 36, we can observe that in most cases, resampling the data brought a positive impact in the scores. When considering the LCPN experiments, there was a positive difference in 11 out of the 13 possible scenarios, with increases in performance of up to 8.96%. For the LCN classifier, there was improvement in 17 out of the 19 scenarios, with increases in performance up to 22.25%.

We also did the same analysis considering the results for each dataset separately. The results of the differences are presented in tables 69 through 72. Considering the LCPN experiments in table 69, resampling had a mixed impact in the results. For three datasets (RYDLS C-19 IDC, ImCLEF07D and Instrument), we found that there were more positive than negative differences. However, for the Actinopterygii, Diptera and iCOPE datasets, there were more negative values than positive ones, which means that in most of the scenarios for these datasets, resampling did not bring any improvements. Our conclusion considering this analysis is that resampling was only effective in some datasets. However, for the LCN experiments in table 71, we had a different result. In these experiments we observed that, in five out of the six datasets positive differences compared to the baseline were reported in most of the scenarios.

Based on these tables 69 and 71, we may summarize the results in terms of which percentage of the scenarios were positive or negative differences. The summary in presented in table 37.

Table 37 – Summary of the baseline difference analysis for each dataset

Result	LCPN		LCN	
	No. Scenarios	%	No. Scenarios	%
Positive	50	64.10	90	78.95
Negative	28	35.90	24	21.05
No Difference	0	0.00	0	0.00
Total	78	100.00	114	100.00

From the summary presented in table 37, we can conclude that, for both LCPN and LCN experiments, we had positive differences in most of the cases. For the LCPN, we had improvement in the scores in 64.10% of the scenarios tested. In the LCN experiments, we found improvements in 78.95% of the scenarios. As a comparison, in section 5.3.1, the results reported scenarios showing statistically significant improvement in 46.15% of the LCPN scenarios and 52.63% of the LCN scenarios. With this information in mind, we can conclude that data resampling improved the macro-averaged F1-scores for the experiments in most scenarios, even if the improvement was not statistically significant. However, to evaluate which strategy is the best ranked one, we propose to apply both Friedman and Nemenyi tests (DEMŠAR, 2006). The Friedman test tells us if there is a significant difference between the groups (in this case represented by the resampling strategies) and if there is, Nemenyi's post-hoc test is used to verify which one had the best performance over the others.

5.3.3 Friedman Test

First, we start by analyzing the LCPN experiments by applying the Friedman test with a significance level of 95% ($p < 0.05$) (DEMŠAR, 2006) and the Nemenyi post-hoc test. The table 38 shows us the average ranks for each strategy and figure 32 illustrates the Nemenyi tests' results through Critical Difference (CD) plots. In these plots, each resampling algorithm is placed in a horizontal axis according to their average ranks, and horizontal lines connects the algorithms that do not have significant difference between them. Lower ranks (more to the left in the plots) means higher macro-averaged F1-Scores.

Looking at table 38 and figure 32 we noticed that the Flat-SMOTE strategy was the best ranked one with an average rank of 6.31, followed very closely by Flat-ROS, Local-SMOTE (both with average ranks of 6.40) and Local-SMOTE-Tomek with an average rank of 6.48. Considering the result from table 29, all of these approaches also showed statistically significant difference when individually compared to the baseline.

Besides analyzing which resampling strategy was the best one for all datasets, we also ran the same analysis for each one of the datasets, and the results are presented in

Table 38 – Average Ranks for each resampling strategy, with a CD = 1.48 and $p = 1.97 \times 10^{-26}$. Results for the LCPN experiments.

Resampling Strategy	Avg. Rank.
flat-smote	6.31
flat-ros	6.40
local-smote	6.40
local-smote-tomek	6.48
local-ros	6.57
flat-smote-tomek	6.66
flat-borderline	6.99
flat-adasyn	7.94
local-selective	7.94
local-borderline	8.02
baseline	8.12
local-adasyn	8.84
flat-smote-enn	9.12
local-smote-enn	9.20

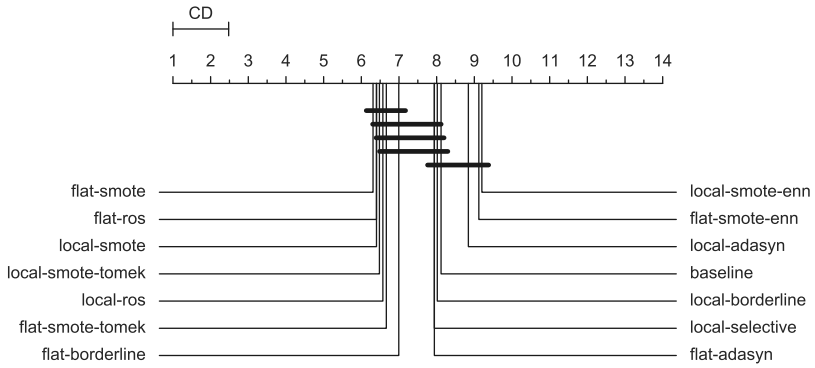


Figure 32 – CD plot from Nemenyi's test ($\alpha = 0.05$), comparing the resampling strategies used with the LCPN Hierarchical classifier

table 39. Based on these results, we also plotted the CD plots depicted in Appendix B in figures 34 through 39. The top 3 results for each dataset are highlighted.

Based on the ranks from table 39 and based on figures 34 through 39 from Appendix B, we can see the reason why the Flat-SMOTE strategy was the best one throughout all datasets. In three out of the six datasets, the Flat-SMOTE strategy was in the top 3 of those datasets.

When the Friedman's test was applied to the results of the LCN experiments, we also obtained a $p < 0.05$, which means that we had significant difference between the groups (resampling strategies). The average ranks, considering the LCN experiments across all datasets are presented in table 40. Using these results, we also plotted the CD plot depicted in figure 33.

Based on these results from table 40 and figure 33, we can conclude that using flat

Table 39 – Average Ranks for each resampling strategy and each dataset separately. Results for the LCPN experiments.

Strategy	Actinopterygii	RYDLS C-19 IDC	Diptera	ImCLEF07D	iCOPE	Instrument
flat-adasyn	6.63	7.53	11.60	7.73	9.27	**4.87**
flat-borderline	**4.90**	8.00	7.33	7.00	8.13	6.60
flat-ros	**4.18**	8.23	7.90	**5.53**	6.47	6.10
flat-smote	6.43	**6.18**	7.03	**5.25**	8.55	**4.42**
flat-smote-enn	11.38	7.73	9.50	7.47	11.10	7.53
flat-smote-tomek	8.02	**5.78**	6.73	**5.35**	9.27	**4.83**
local-adasyn	9.80	8.10	9.60	10.20	6.72	8.63
local-borderline	7.87	**6.03**	8.43	9.23	6.67	9.87
local-ros	**4.78**	6.87	6.90	6.33	**5.02**	9.53
local-selective	8.77	8.30	6.60	9.23	6.17	8.57
local-smote	7.18	6.87	**5.33**	6.67	**6.13**	6.23
local-smote-enn	10.83	8.70	8.70	8.63	8.83	9.50
local-smote-tomek	7.62	6.27	**4.70**	7.23	**5.73**	7.33
baseline	6.60	10.40	**4.63**	9.13	6.95	10.98

Table 40 – Average Ranks for each resampling strategy with a CD = 2.21 and $p = 3.00 \times 10^{-163}$, considering the LCN experiments.

Resampling Strategy	Avg. Rank.
threshold-smote-tomek	6.34
threshold-smote	6.43
local-smote	6.86
threshold-smote-enn	7.36
local-smote-tomek	7.43
threshold-ros	8.26
local-ros	9.24
threshold-adasyn	9.32
local-smote-enn	9.51
local-selective	9.75
threshold-borderline	10.06
local-adasyn	10.81
local-borderline	11.27
baseline	12.96
flat-smote-tomek	13.27
flat-smote	13.79
flat-ros	13.94
flat-borderline	14.19
flat-smote-enn	14.44
flat-adasyn	14.78

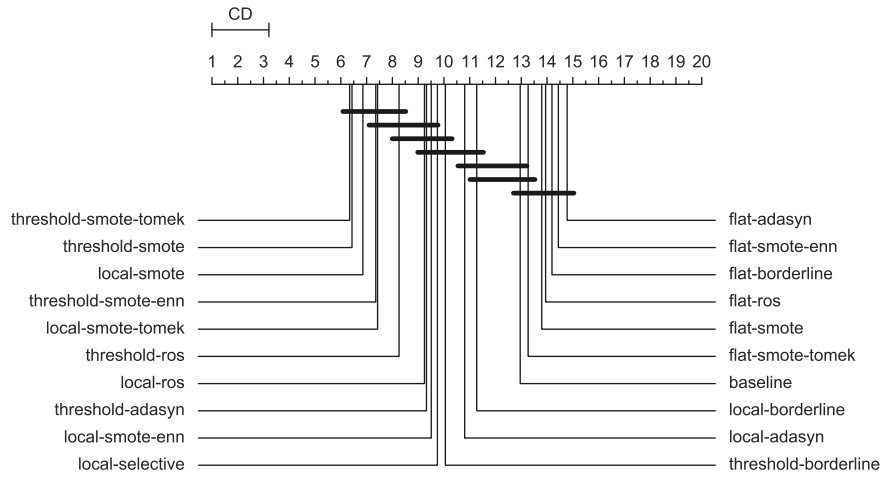


Figure 33 – CD plot from Nemenyi's test ($\alpha = 0.05$), comparing the resampling strategies used with the LCN Hierarchical classifier

resampling in the LCN experiments was not effective at all, since their performance was below the baseline. Also, we noticed that the Threshold Selective resampling approach, which was proposed in this work, was the best ranked one when used together with the SMOTE-Tomek algorithm. The Threshold Selective approach used together with the SMOTE algorithm also had good results, having a performance that was very close to the Threshold-SMOTE-Tomek strategy.

In order to have a more detailed look at these results, we also ran the same statistical tests separately for each of the six datasets. The average ranks obtained in these tests are presented in table 41. The CD plots for each one of them were also created, and they are presented in the Appendix B, in figures 40 through 45. The top 3 ranked strategies for each dataset are highlighed in table 41.

From table 41 and figures 40 to 45 in Appendix B, we can observe an interesting trend in the average ranks. In five out of the six datasets, the Threshold-SMOTE-Tomek strategy was in the top 3 approaches for these datasets. Also, the Threshold-SMOTE strategy was in the top 3 in four out of the six datasets, which is consistent with our previous analysis where we considered all datasets, in table 40. Another trend that we observed is that, most of the best results for each dataset are concentrated in the Threshold Selective resampling approach. The only outlier was the iCOPE dataset, which did not have good ranks using the Threshold Selective approach and performed better with the Local resampling approach. With that in mind, we can conclude that the proposed approach, Threshold Selective resampling, was the best one for the LCN experiments when used together with the SMOTE-Tomek algorithm.

Table 41 – Average Ranks for each resampling strategy and each dataset separately, considering the LCN scenario.

Strategy	Actinopterygii	RYDLS C-19 IDC	Diptera	ImCLEF07D	iCOPE	Instrument
flat-adasyn	15.00	13.90	18.40	16.00	10.05	15.33
flat-borderline	11.47	14.67	16.23	16.27	10.65	15.87
flat-ros	13.00	12.87	16.80	12.80	11.83	16.37
flat-smote	13.93	13.07	16.07	13.63	11.40	14.67
flat-smote-enn	16.30	13.53	10.63	13.30	16.77	16.10
flat-smote-tomek	14.13	12.40	13.80	13.70	11.77	13.80
local-adasyn	10.20	10.67	11.90	12.73	9.13	10.20
local-borderline	10.97	10.45	13.17	13.50	9.82	9.73
local-ros	10.97	9.15	10.83	6.90	**8.98**	8.63
local-selective	9.63	10.53	7.23	12.77	**8.12**	10.20
local-smote	8.50	**6.48**	4.80	7.50	**9.02**	**4.83**
local-smote-enn	12.57	9.65	7.33	6.87	11.23	9.40
local-smote-tomek	8.73	8.18	5.00	8.00	9.60	5.03
baseline	14.80	9.67	14.53	11.07	10.45	17.23
threshold-adasyn	6.73	9.97	10.03	10.43	9.80	8.97
threshold-borderline	7.83	9.98	11.37	10.47	11.32	9.37
threshold-ros	8.53	8.52	9.03	**5.77**	9.57	8.13
threshold-smote	**5.47**	**7.22**	**4.00**	6.43	10.65	**4.83**
threshold-smote-enn	**6.43**	10.97	**4.50**	**5.60**	9.30	7.33
threshold-smote-tomek	**4.80**	**8.13**	**4.33**	**6.27**	10.55	**3.97**

6 Concluding Remarks

Imbalanced data is an issue that may affect data distribution across many different domains. It is a known issue in classification tasks, and it is usually associated with rare patterns, which could be a kind of data that is difficult to collect or it is just a kind of observation that does not happen very often. When we model a classification problem, these rare patterns are usually known as the minority classes while the most frequent ones are the known as the majority classes. The class imbalance affects different kinds of classification problems as well. From simple binary classification, like in the credit card fraud detection, to more complex ones, like hierarchical classification problems.

This imbalanceness in the class distribution usually causes the classifiers to be biased towards the majority classes, which sometimes is not good to the problem we are evaluating. In some domains, like medical imaging, where the minority classes are often associated with diseases, we are a lot more interested in detecting the rare patterns. Another example would be the credit card fraud detection, where the fraudulent transactions are also patterns that do not occur very often, but they are the ones we want to be able to identify. For these kind of classification problems, where we are more interested in the minority classes, dealing with class imbalance is a very important part of the process to develop and train an accurate classifier.

There are multiple ways to deal with class imbalance in the literature, and the most versatile one is the data level approach, where we use resampling algorithms to generate new data samples in order to make the dataset balanced (LOYOLA-GONZÁLEZ et al., 2016). We proposed in this work to use different resampling approaches and algorithms, adapting it to a hierarchical classification task.

As part of this work, we proposed to investigate and evaluate the use of data resampling in Local Hierarchical Classification. First, we proposed to establish a baseline where no data resampling is used and then compare the results with different resampling approaches, such as the Flat, Local, Threshold Selective and Local Selective approaches mentioned in section 4.4. It is important to note that in this work we proposed two new approaches to resample the data: the Threshold Selective and the Local Selective. For all the resampling approaches, we used different resampling algorithms available in the literature.

After gathering the results for the baseline and for each of the proposed resampling approaches, we analyzed the results using non-parametric statistical tests in order to prove with statistical significance if there were statistically significant improvements in the classification results of those experiments where resampling was used.

We divided our analysis in three steps: first, we evaluated each resampling strategy individually against the results obtained with the baseline using a Wilcoxon Signed-Rank (WILCOXON, 1945) test for paired samples, in order to verify if there was statistically significant improvement when using resampling, compared to the baseline. In a second moment, in order to cover those cases where there was no statistically significant difference, we simply calculated the difference in the scores, subtracting the scores of each resampling approach and the baseline. Positive results indicated the experiments where there was any improvement and negative results indicated those where the performance decreased. Lastly, we analyzed which of the approaches were the best ones by using the Friedman and Nemenyi tests (DEMŠAR, 2006).

From the results presented in tables 29 through 33, we can observe that for both LCPN and LCN hierarchical classification approaches, there were resampling strategies that yielded statistically significant improvements. If we observe the results that considered all datasets in tables 29 and 32, we may notice that, out of the 13 strategies tested, 9 yielded improvements for the LCPN classifier. For the LCN classifier, from the 19 strategies tested, 13 of them yielded statistically significant improvements. Based on this analysis, and in our hypothesis stated in section 1.2, we may accept the hypothesis that that resampling the data in local hierarchical classification problems brings statistically significant improvements to the classification results.

However, if we take a deeper look at the results for each dataset separately in tables 30 and 33, we may notice that this is not true for every dataset. Resampling the data was more effective in some datasets than others. For the LCPN classifier, we observed statistically significant improvement in 36 out of the 78 scenarios (46.15%). In the Diptera dataset, no improvements in any scenario were found. For the LCN classifier, we noticed statistically significant improvement in 60 out of the 114 scenarios (52.63%). In this case, we did not find any improvement when using the RYDLS C-19 IDC and iCOPE datasets.

In the second step of the analysis, we analyzed the difference between the baseline and the resampling approaches by doing the subtraction between the score reported with each approach and the one obtained in the baseline. Our objective with this analysis was finding out all the scenarios that any improvement was recorded, even if it was not statistically significant. From tables 34 and 36, where the analysis was performed across all datasets, it is possible to observe that in most cases, data resampling resulted in positive differences, which means that it improved the classification performance. In the LCPN experiments, there was a positive difference in 11 out of the 13 scenarios while in the LCN experiments there was a positive difference in 17 out of the 19 scenarios.

In the separate analysis performed for each dataset, summarized in table 37, it is possible to notice that data resampling yielded positive differences in most of the scenarios as well. For the LCPN experiments, there was improvement in 64.10% of the scenarios,

while in the LCN experiments there was improvement in 78.95% of the scenarios.

In the third and last step of our analysis, we analyzed which resampling strategies were the best ones with the average rank results presented in tables 38 through 41 and the CD plots in figures 32 and 33 and also in in Appendix B, in figures 34 through 45.

For the LCPN experiments, it is possible to observe that the best ranked resampling strategy considering all datasets was the Flat-SMOTE strategy with an average rank of 6.31, as presented in table 38. When analyzing the performance for each dataset separately, we noticed that the Flat-SMOTE appeared in the top 3 ranked approaches of each dataset in three out of the six datasets. For the LCN experiments, based on the average ranks from table 40 we noticed that the Flat Resampling approach was not effective in any scenarios. From these results, it is also possible to notice that, one of the resampling approaches proposed in this work, the Threshold Selective Resampling was the best ranked one when used together with the SMOTE-Tomek algorithm (Threshold-SMOTE-Tomek approach), with an average rank of 6.34. By looking at the results presented in table 41 for each dataset separately, we noticed that the Threshold-SMOTE-Tomek approach appears in the top 3 results for each dataset in five out of the six datasets. The other resampling approach proposed in this work, called Local Selective Resampling, was not among the best ranked ones and because of this we may conclude that it still needs adjustments on its methodology.

Based on this analysis and in the objectives defined in 1.1, it is possible to conclude that, after comparing the results where data resampling was used with those obtained for the baseline, data resampling in local hierarchical classification yields statistically significant improvements in some scenarios, and an increase in the classification scores in most of the scenarios. It was also possible to conclude that one of the proposed resampling approaches, the Threshold Selective, was effective and yielded the best average ranks for the LCN experiments. Because of this, we may conclude that the Threshold-Selective resampling approach may be a good solution to mitigate losing performance in the classification of majority classes while still resampling the minority ones. On the other hand, we also concluded that the other approach proposed in this work, called Local Selective, still needs some amendments in order to produce better results.

Some of the limitations of this work are: only MLNP and tree-based problems were investigated, and a limited number of resampling approaches were tested. As future research direction, we suggest investigating NMLNP and graph-based problems as well as conducting further investigation in other resampling approaches.

Bibliography

ABAD, Zahra Shakeri Hossein; MASLOVE, David; LEE, Joon. Predicting discharge destination of critically ill patients using machine learning. *IEEE Journal of Biomedical and Health Informatics*, IEEE, 2020. Referenced 3 times in pages 44, 45, and 46.

ABDULRAZZAQ, M. M.; YASEEN, I. F. T.; NOAH, S. A.; FADHIL, M. A.; ASHOUR, M. U. Xmiar: X-ray medical image annotation and retrieval. In: ARAI, Kohei; KAPOOR, Supriya (Ed.). *Advances in Computer Vision*. Cham: Springer International Publishing, 2020. p. 638–651. ISBN 978-3-030-17798-0. Referenced in page 16.

ADDI, Hajar Ait; EZZAHIR, Redouane; MAHMOUDI, Abdelhak. Three-level binary tree structure for sentiment classification in Arabic text. *ACM International Conference Proceeding Series*, 2020. Referenced 4 times in pages 44, 45, 46, and 47.

ARIAS, Jacinto; MARTÍNEZ-GÓMEZ, Jesus; GÁMEZ, Jose A.; Seco de Herrera, Alba G.; MÜLLER, Henning. Medical image modality classification using discrete Bayesian networks. *Computer Vision and Image Understanding*, v. 151, p. 61–71, 2016. ISSN 1090235X. Referenced in page 16.

BAI, Jie; JIANG, Huiyan; LI, Siqi; MA, Xiaoqi. NHL Pathological Image Classification Based on Hierarchical Local Information and GoogLeNet-Based Representations. *BioMed Research International*, v. 2019, 2019. ISSN 23146141. Referenced in page 16.

BARROS, Fábio K. H. de; SELLETI, André L. Jeller; QUEIROZ, Vinícius A. P.; PEREIRA, Rodolfo M.; SILLA, Carlos N. Analyzing the impact of resampling approaches on chest x-ray images for covid-19 identification in a local hierarchical classification scenario. In: *Proceedings of the The* 21^{st} *IEEE International Conference on BioInformatics and BioEngineering*. Kragujevac, Serbia: IEEE, 2021. Referenced 2 times in pages 18 and 75.

BATISTA, Gustavo E. A. P. A.; PRATI, Ronaldo C.; MONARD, Maria Carolina. A study of the behavior of several methods for balancing machine learning training data. *ACM SIGKDD Explorations Newsletter*, v. 6, n. 1, p. 20–29, 2004. ISSN 1931-0145. Referenced 3 times in pages 35, 41, and 42.

CHAWLA, Nitesh V.; BOWYER, Kevin W.; HALL, Lawrence O.; KEGELMEYER, W. Philip. SMOTE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 2002. ISSN 10769757. Referenced in page 36.

COHEN, Joseph Paul; MORRISON, Paul; DAO, Lan; ROTH, Karsten; DUONG, Tim Q; GHASSEMI, Marzyeh. COVID-19 Image Data Collection: Prospective Predictions Are the Future. v. 2019, p. 1–38, 2020. Disponível em: <http://arxiv.org/abs/2006.11988>. Referenced 3 times in pages 22, 49, and 50.

COLUMBUS, Louis. *Roundup Of Machine Learning Forecasts And Market Estimates, 2020*. 2020. Disponível em: <https://www.forbes.com/sites/louiscolumbus/2020/01/19/roundup-of-machine-learning-forecasts-and-market-estimates-2020/?sh=3bb28e415c02>. Referenced in page 16.

DEMŠAR, Janez. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, v. 7, n. 1, p. 1–30, 2006. Disponível em: <http://jmlr.org/papers/v7/demsar06a.html>. Referenced 4 times in pages 74, 81, 87, and 93.

DIMITROVSKI, Ivica; KOCEV, Dragi; LOSKOVSKA, Suzana; DŽEROSKI, Sašo. Hierarchical annotation of medical images. *Pattern Recognition*, v. 44, n. 10-11, p. 2436–2449, 2011. ISSN 00313203. Referenced 2 times in pages 48 and 49.

DUMAIS, Susan; CHEN, Hao. Hierarchical classification of web content. In: *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*. [S.l.: s.n.], 2000. p. 256–263. Referenced in page 30.

FELIPE, Gustavo Z; AGUIAR, Rafael L; COSTA, Yandre MG; SILLA, Carlos N; BRAHNAM, Sheryl; NANNI, Loris; MCMURTREY, Shannon. Identification of infants' cry motivation using spectrograms. In: IEEE. *2019 International Conference on Systems, Signals and Image Processing (IWSSIP)*. [S.l.], 2019. p. 181–186. Referenced 2 times in pages 48 and 49.

FENG, Shou; FU, Ping; ZHENG, Wenbin. A Hierarchical multi-label classification algorithm for gene function prediction. *Algorithms*, v. 10, n. 4, p. 1–14, 2017. ISSN 19994893. Referenced 4 times in pages 44, 45, 46, and 73.

FENG, Shou; FU, Ping; ZHENG, Wenbin. A postprocessing method in the HMC framework for predicting gene function based on biological instrumental data. *Review of Scientific Instruments*, v. 89, n. 3, 2018. ISSN 10897623. Disponível em: <http://dx.doi.org/10.1063/1.5010353>. Referenced 3 times in pages 44, 45, and 46.

FERNÁNDEZ, Alberto; LÓPEZ, Victoria; GALAR, Mikel; JESUS, MaríA José Del; HERRERA, Francisco. Analysing the classification of imbalanced data-sets with multiple classes: Binarization techniques and ad-hoc approaches. *Knowledge-based systems*, Elsevier, v. 42, p. 97–110, 2013. Referenced 3 times in pages 16, 33, and 44.

HAN, Hui; WANG, Wen-yuan; MAO, Bing-huan. Borderline-SMOTE: A New Over-Sampling Method in Imbalanced Data Sets Learning. In: *International Conference on Intelligent Computing, ICIC 2005, Hefei, China, August 23-26, 2005, Proceedings, Part I*. [S.l.: s.n.], 2005. ISBN 978-3-540-31902-3. Referenced in page 38.

HE, Haibo; BAI, Yang; GARCIA, Edwardo A.; LI, Shutao. ADASYN: Adaptive synthetic sampling approach for imbalanced learning. *Proceedings of the International Joint Conference on Neural Networks*, n. 3, p. 1322–1328, 2008. Referenced in page 39.

HOSENIE, Zafiirah; LYON, Robert J.; STAPPERS, Benjamin W.; MOOTOOVALOO, Arrykrishna. Comparing multiclass, binary, and hierarchical machine learning classification schemes for variable stars. *Monthly Notices of the Royal Astronomical Society*, Oxford University Press, v. 488, n. 4, p. 4858–4872, 2019. ISSN 13652966. Referenced 2 times in pages 45 and 46.

KIRITCHENKO, Svetlana; MATWIN, Stan; FAMILI, A Fazel et al. Functional annotation of genes using hierarchical text categorization. In: *Proc. of the ACL Workshop on Linking Biological Literature, Ontologies and Databases: Mining Biological Semantics*. [S.l.: s.n.], 2005. Referenced in page 30.

KIRITCHENKO, Svetlana; MATWIN, Stan; NOCK, Richard; FAMILI, A. Fazel. Learning and evaluation in the presence of class hierarchies: Application to text categorization. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. [S.l.: s.n.], 2006. ISBN 3540346287. ISSN 16113349. Referenced 3 times in pages 30, 73, and 74.

KLUNGPORNKUN, Mongkud; VATEEKUL, Peerapon. Hierarchical text categorization using level based neural networks of word embedding sequences with sharing layer information. *Walailak Journal of Science and Technology*, v. 16, n. 2, p. 121–131, 2019. ISSN 2228835X. Referenced 3 times in pages 44, 45, and 46.

KOHAVI, Ron. A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. *International Joint Conference of Artificial Intelligence*, n. March 2001, 1995. Referenced 2 times in pages 50 and 76.

KOLLER, Daphne; SAHAMI, Mehran. Hierarchically classifying documents using very few words. In: *14th International Conference on Machine Learning*. [S.l.: s.n.], 1997. Referenced in page 25.

KUMAR, Sanjiv; ROWLEY, Henry Allan; WANG, Xiaohang; RODRIGUES, Jose Jeronimo Moreira. *Hierarchical classification in credit card data extraction*. [S.l.]: Google Patents, 2015. US Patent 9,213,907. Referenced in page 16.

LOYOLA-GONZÁLEZ, Octavio; MARTÍNEZ-TRINIDAD, José Fco; CARRASCO-OCHOA, Jesús Ariel; GARCÍA-BORROTO, Milton. Study of the impact of resampling methods for contrast pattern based classifiers in imbalanced databases. *Neurocomputing*, Elsevier, v. 175, p. 935–947, 2016. ISSN 18728286. Disponível em: <http://dx.doi.org/10.1016/j.neucom.2015.04.120>. Referenced 2 times in pages 17 and 92.

NAKANO, Felipe Kenji; LIETAERT, Mathias; VENS, Celine. Machine learning for discovering missing or wrong protein function annotations. *BMC bioinformatics*, Springer, v. 20, n. 1, p. 1–32, 2019. Referenced in page 47.

ORRIOLS-PUIG, Albert; BERNADÓ-MANSILLA, Ester. Evolutionary rule-based systems for imbalanced data sets. *Soft Computing*, Springer, v. 13, n. 3, p. 213–225, 2009. Referenced 2 times in pages 32 and 61.

PARMEZAN, Antonio Rafael Sabino; SOUZA, Vinicius M. A.; BATISTA, Gustavo E. A. P. A. Towards hierarchical classification of data streams. In: VERA-RODRIGUEZ, Ruben; FIERREZ, Julian; MORALES, Aythami (Ed.). *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*. Cham: Springer International Publishing, 2019. p. 314–322. ISBN 978-3-030-13469-3. Referenced 2 times in pages 48 and 49.

PARMIGIANI, G. Decision theory: Bayesian. In: SMELSER, Neil J.; BALTES, Paul B. (Ed.). *International Encyclopedia of the Social & Behavioral Sciences*. Oxford: Pergamon, 2001. p. 3327–3334. ISBN 978-0-08-043076-8. Disponível em: <https://www.sciencedirect.com/science/article/pii/B0080430767004034>. Referenced in page 16.

PEREIRA, Rodolfo M.; BERTOLINI, Diego; TEIXEIRA, Lucas O.; SILLA, Carlos N.; COSTA, Yandre M.G. COVID-19 identification in chest X-ray images on flat and

hierarchical classification scenarios. *Computer Methods and Programs in Biomedicine*, Elsevier B.V., v. 194, 2020. ISSN 18727565. Referenced 7 times in pages 16, 22, 45, 46, 47, 49, and 50.

PEREIRA, Rodolfo M; COSTA, Yandre MG; SILLA, Carlos N. Handling imbalance in hierarchical classification problems using local classifiers approaches. *Data Mining and Knowledge Discovery*, Springer, p. 1–58, 2021. Referenced 3 times in pages 45, 46, and 47.

PEREIRA, Rodolfo Miranda; COSTA, Yandre Maldonado E Gomes Da; SILLA, Carlos Nascimento. Dealing with imbalanceness in hierarchical multi-label datasets using multi-label resampling techniques. *Proceedings - International Conference on Tools with Artificial Intelligence, ICTAI*, v. 2018-November, p. 818–824, 2018. ISSN 10823409. Referenced 3 times in pages 45, 46, and 47.

RACHMAN, G. H.; KHODRA, M. L.; WIDYANTORO, D. H. Hierarchical Rhetorical Sentence Categorization for Scientific Papers. *Journal of Physics: Conference Series*, v. 978, n. 1, 2018. ISSN 17426596. Referenced 3 times in pages 44, 45, and 46.

SAHARE, Mahendra; GUPTA, Hitesh. A review of multi-class classification for imbalanced data. *International Journal of Advanced Computer Research*, Citeseer, v. 2, n. 3, p. 160, 2012. Referenced in page 16.

Secretaria de Desenvolvimento Econômico. *Desafio: Como o uso de algoritmos de inteligência artificial pode auxiliar médicos radiologistas no diagnóstico do Covid-19 através de imagem de Tomografia Computadorizada e Raios-x de tórax?* [S.l.]: Governo de São Paulo, 2020. <https://ideiagov.sp.gov.br/wp-content/uploads/2020/07/relatorio-final-imagem-tomo-e-raio-x.pdf>. Referenced in page 18.

SILLA, Carlos N.; FREITAS, Alex A. A survey of hierarchical classification across different application domains. *Data Mining and Knowledge Discovery*, v. 22, n. 1-2, p. 31–72, 2011. ISSN 13845810. Referenced 8 times in pages 16, 20, 21, 23, 26, 30, 48, and 67.

SILVA-PALACIOS, Daniel; FERRI, Cèsar; RAMÍREZ-QUINTANA, María José. Improving Performance of Multiclass Classification by Inducing Class Hierarchies. *Procedia Computer Science*, v. 108, p. 1692–1701, 2017. ISSN 18770509. Referenced 2 times in pages 45 and 46.

SUN, Aixin; LIM, E-P; NG, W-K; SRIVASTAVA, Jaideep. Blocking reduction strategies in hierarchical text classification. *IEEE Transactions on Knowledge and Data Engineering*, IEEE, v. 16, n. 10, p. 1305–1308, 2004. Referenced in page 30.

TOMEK, Ivan et al. Two modifications of cnn. 1976. Referenced in page 41.

TSOUMAKAS, Grigorios; KATAKIS, Ioannis. Multi-label classification: An overview. *International Journal of Data Warehousing and Mining (IJDWM)*, IGI Global, v. 3, n. 3, p. 1–13, 2007. Referenced in page 16.

VATEEKUL, Peerapon; KUBAT, Miroslav; SARINNAPAKORN, Kanoksri. Hierarchical multi-label classification with SVMs: A case study in gene function prediction. *Intelligent Data Analysis*, v. 18, n. 4, p. 717–738, 2014. ISSN 15714128. Referenced in page 73.

WILCOXON, Frank. Individual comparisons by ranking methods. *Biometrics Bulletin*, JSTOR, v. 1, n. 6, p. 80–83, 1945. Referenced 3 times in pages 74, 81, and 93.

WILSON, Dennis L. Asymptotic properties of nearest neighbor rules using edited data. *IEEE Transactions on Systems, Man, and Cybernetics*, IEEE, n. 3, p. 408–421, 1972. Referenced in page 41.

Appendix

APPENDIX A – Complete class distribution of all datasets

Table 42 – Class Distribution of the Actinopterygii dataset

Class	Count
R/perciformes/pomacentridae/chromis/c-chrysura	3593
R/perciformes/pomacentridae/dascyllus/d-reticulatus	3196
R/beryciformes/holocentridae/myripristis/m-kuntee	3004
R/perciformes/pomacentridae/amphiprion/a-clarkii	2985
R/perciformes/acanthuridae/acanthurus/a-nigrofuscus	2511
R/perciformes/chaetodontidae/chaetodon/c-lunulatus	2494
R/perciformes/pomacentridae/plectroglyphidodon/p-dickii	2456
R/perciformes/pomacentridae/dascyllus/d-aruanus	904
R/perciformes/chaetodontidae/chaetodon/c-trifascialis	375
R/perciformes/pomacentridae/abudefduf/a-vaigiensis	306
R/perciformes/acanthuridae/zebrasoma/z-scopas	271
R/perciformes/labridae/hemigymnus/h-melapterus	147
R/perciformes/pomacentridae/neoglyphidodon/n-nigroris	129
R/perciformes/pempheridae/pempheris/p-vanicolensis	49
R/perciformes/chaetodontidae/chaetodon/c-speculum	24

Table 43 – Class Distribution of the Diptera dataset

Class	Count
R/drosophilidae/drosophila/d-melanogaster	2005
R/muscidae/musca/m-domestica	2005
R/drosophilidae/drosophila/d-suzukii	2005
R/chironomidae/chironomus/c-xanthus	2005
R/culicidae/anopheles/a-aquasalis/a-aquasalis:f	1505
R/culicidae/aedes/ae-albopictus/ae-albopictus:f	1505
R/culicidae/culex/cx-quinquefasciatus/cx-quinquefasciatus:m	1505
R/culicidae/aedes/ae-albopictus/ae-albopictus:m	1505
R/culicidae/culex/cx-quinquefasciatus/cx-quinquefasciatus:f	1505
R/culicidae/culex/cx-tarsalis/cx-tarsalis:f	1505
R/culicidae/anopheles/a-aquasalis/a-aquasalis:m	1505
R/culicidae/aedes/ae-aegypti/ae-aegypti:f	1505
R/culicidae/aedes/ae-aegypti/ae-aegypti:m	1505
R/culicidae/culex/cx-tarsalis/cx-tarsalis:m	157

Table 44 – Class Distribution of the Instrument dataset

Class	Count
R/aerophone/wood/flute	1029
R/aerophone/brass/tuba	1009
R/aerophone/wood/clarinet	977
R/aerophone/wood/saxophone	860
R/aerophone/brass/trombone	823
R/aerophone/wood/bassoon	760
R/aerophone/brass/horn	696
R/aerophone/wood/oboe	631
R/aerophone/brass/trumpet	556
R/idiophone/fibrousmaterial/marimba	303
R/membranophone/bimembranophone/snare	198
R/idiophone/fibrousmaterial/xylophone	185
R/chordophone/pizzicato/guitar	106
R/chordophone/pizzicato/doublebass	104
R/eletrophone/electronic/synthesizer	103
R/membranophone/unimembranophone/tambourine	102
R/chordophone/arco/violadarco	100
R/chordophone/arco/cello	95
R/membranophone/bimembranophone/bassdrum	94
R/chordophone/arco/violin	90
R/chordophone/pizzicato/mandolin	80
R/idiophone/metal/cymbal	77
R/chordophone/pizzicato/banjo	74
R/eletrophone/electroacoustic/electricguitar	73
R/eletrophone/electronic/keyboard	58
R/membranophone/unimembranophone/timbale	53
R/membranophone/bimembranophone/toms	45
R/eletrophone/electroacoustic/bass	44
R/idiophone/metal/bells	41
R/idiophone/metal/gong	28
R/idiophone/metal/crotale	25

Table 45 – Class Distribution of the ImCLEF07D dataset

Class	Count
R/1/2/0	2788
R/1/2/7	2166
R/1/1/0	1704
R/2/1/1	952
R/2/1/0	642
R/2/3/0	605
R/2/4/0	322
R/2/2/0	315
R/4/a/0	173
R/1/1/5	172
R/4/1/0	151
R/3/1/0	148
R/2/0/0	140
R/4/9/0	131
R/4/3/0	114
R/3/2/0	107
R/4/2/0	94
R/1/2/1	86
R/1/1/2	51
R/4/b/0	41
R/1/2/f	32
R/4/6/0	17
R/1/1/6	16
R/1/2/9	15
R/2/2/8	12
R/2/2/9	12

Table 46 – Class Distribution of the iCOPE dataset

Class	Count
R/Pain	42
R/NoPain/Rest	36
R/NoPain/Friction	20
R/NoPain/Movement	15

Table 47 – Class Distribution of the RYDLS C-19 IDC dataset

Class	Count
R/Normals	1000
R/Pneumonia/Viral/COVID-19	476
R/Pneumonia/Fungal/Pneumocystis	24
R/Pneumonia/Bacterial/Streptococcus	18
R/Pneumonia/Viral/SARS	16
R/Tuberculosis	11
R/Pneumonia/Viral/MERS-CoV	10
R/Pneumonia/Bacterial/Legionella	9
R/Pneumonia/Bacterial/Klebsiella	9
R/Pneumonia/Lipoid	8

Table 48 – Hierarchical Class Distribution of the Actinopterygii dataset

1st level		2nd level		3rd level		4th level	
perciformes	19440	pomacentridae	13569	dascyllus	4100	d-reticulatus	3196
						d-aruanus	904
				chromis	3593	c-chrysura	3593
				amphiprion	2985	a-clarkii	2985
				plectroglyphidodon	2456	p-dickii	2456
				abudefduf	306	a-vaigiensis	306
				neoglyphidodon	129	n-nigroris	129
		acanthuridae	2782	acanthurus	2511	a-nigrofuscus	2511
				zebrasoma	271	z-scopas	271
		chaetodontidae	2893	chaetodon	2893	c-lunulatus	2494
						c-trifascialis	375
						c-speculum	24
		labridae	147	hemigymnus	147	h-melapterus	147
		pempheridae	49	pempheris	49	p-vanicolensis	49
beryciformes	3004	holocentridae	3004	myripristis	3004	m-kuntee	3004

Table 49 – Hierarchical Class Distribution of the Diptera dataset

1st level		2nd level		3rd level		4th level	
drosophilidae	4010	drosophila	4010	d-melanogaster	2005	–	–
				d-suzukii	2005	n/a	n/a
muscidae	2005	muscidae	2005	musca	2005	m-domestica	2005
chironomidae	2005	chironomidae	2005	chironomus	2005	c-xanthus	2005
culicidae	13702	aedes	6020	ae-albopictus	3010	ae-albopictus:m	1505
						ae-albopictus:f	1505
				ae-aegypti	3010	ae-aegypti:f	1505
						ae-aegypti:m	1505
		culex	4672	cx-quinquefasciatus	3010	cx-quinquefasciatus:m	1505
						cx-quinquefasciatus:f	1505
				cx-tarsalis	1662	cx-tarsalis:f	1505
						cx-tarsalis:m	157
		anopheles	3010	a-aquasalis	3010	a-aquasalis:f	1505
						a-aquasalis:m	1505

Table 50 – Hierarchical Class Distribution of the Instrument dataset

1st level		2nd level		3rd level	
aerophone	7341	wood	4257	flute	1029
				clarinet	977
				saxophone	860
				bassoon	760
				oboe	631
		brass	3084	tuba	1009
				trombone	823
				horn	696
				trumpet	556
chordophone	649	pizzicato	364	guitar	106
				doublebass	104
				mandolin	80
				banjo	74
		arco	285	violadarco	100
				cello	95
				violin	90
eletrophone	278	electroacoustic	117	bass	44
				electricguitar	73
		electronic	161	keyboard	58
				synthesizer	103
membranophone	492	bimembranophone	337	snare	198
				bassdrum	94
				toms	45
		unimembranophone	155	tambourine	102
				timbale	53
idiophone	659	fibrousmaterial	488	marimba	303
				xylophone	185
		metal	171	cymbal	77
				bells	41
				gong	28
				crotale	25

Table 51 – Hierarchical Class Distribution of the ImCLEF07D dataset

1st level	2nd level	3rd level
1 7030	1 1943	0 1704
		5 172
		2 51
		6 16
	2 5087	0 2788
		7 2166
		1 86
		f 32
		9 15
2 3000	0 140	0 140
	1 1594	1 952
		0 642
	2 339	0 315
		8 12
		9 12
	3 605	0 605
	4 322	0 322
3 255	1 148	0 148
	2 107	0 107
4 721	1 151	0 151
	2 94	0 94
	3 114	0 114
	6 17	0 17
	9 131	0 131
	a 173	0 173
	b 41	0 41

Table 52 – Hierarchical Class Distribution of the iCOPE dataset

1st level	2nd level
Pain 42	–
No Pain 71	Rest 36
	Friction 20
	Movement 15

Table 53 – Hierarchical Class Distribution of the RYDLS C-19 IDC dataset

1st level	2nd level	3rd level
Normals 1000	– –	– –
Pneumonia 570	Viral 502	COVID-19 476
		SARS 16
		MERS-CoV 10
	Bacterial 36	Legionella 9
		Klebsiella 9
		Streptococcus 18
	Fungal 24	Pneumocystis 24
	Lipoid 8	– –
Tuberculosis 11	– –	– –

APPENDIX B – Experimental setup and results

Table 54 – Summary of the classification algorithms and the configurations used in the experiments

Algorithm	**Parameters**
Decision Trees (DT)	criterion='gini' splitter='best' max_depth = None min_samples_split=2 min_samples_leaf=1 min_weight_fraction_leaf=0.0 max_features=None max_leaf_nodes=None min_impurity_decrease=0.0 class_weight=None
Random Forest (RF)	n_estimators=150 criterion='gini' max_depth=None min_samples_split=2 min_samples_leaf=1 min_weight_fraction_leaf=0.0 max_features='auto' max_leaf_nodes=None min_impurity_decrease=0.0 bootstrap=True oob_score=False class_weight=None ccp_alpha=0.0 max_samples=None
Multilayer Perceptron (MLP)	hidden_layer_sizes = (100,) solver='lbfgs' activation='relu' alpha=0.0001 learning_rate='constant' learning_rate_init=0.001 max_iter=1000 shuffle=True tol=1e-4 momentum=0.9 early_stopping=False max_fun=15000
SupportVector Machine (SVM)	SVC kernel='rbf' C=1.0 degree=3 gamma='scale' coef0=0.0 tol=1e-3 class_weight=None max_iter=-1 decision_function_shape='ovr' break_ties=False probability=True
Naive Bayes (NB)	GaussianNB() var_smoothing=1e-9
K-Nearest Neighbors (kNN)	n_neighbors=5 weights='uniform' algorithm='auto' leaf_size=30 p=2 metric='minkowski' metric_params=None

Table 55 – Mean macro-averaged F1 core for the LCPN experiments, considering all datasets

Strategy	DT	kNN	MLP	NB	RF	SVM	Avg.
baseline	0.389	0.409	0.287	0.260	0.459	0.271	0.346
flat-adasyn	0.357	0.439	0.313	0.247	0.506	0.321	0.364
flat-borderline	0.365	0.440	0.291	0.251	0.499	0.349	0.366
flat-ros	0.393	0.455	0.275	0.253	0.500	0.356	0.372
flat-smote	0.373	0.432	0.289	0.257	0.524	0.348	0.371
flat-smote-enn	0.339	0.405	0.251	0.249	0.449	0.319	0.335
flat-smote-tomek	0.388	0.438	0.287	0.259	0.502	0.333	0.368
local-adasyn	0.372	0.423	0.285	0.237	0.511	0.316	0.357
local-borderline	0.380	0.424	0.291	0.239	0.494	0.348	0.363
local-ros	0.389	0.445	0.316	0.257	0.493	0.361	0.377
local-smote	0.385	0.432	0.287	0.254	0.517	0.358	0.372
local-smote-enn	0.358	0.381	0.280	0.252	0.459	0.312	0.340
local-smote-tomek	0.394	0.426	0.294	0.259	0.510	0.355	0.373
local-selective	0.376	0.417	0.355	0.259	0.483	0.348	0.373

Table 56 – Mean macro-averaged F1 score for the LCPN experiments, for the Actinopterygii dataset

Strategy	DT	kNN	MLP	NB	RF	SVM	Avg.
baseline	0.551	0.360	0.114	0.175	0.659	0.045	0.317
flat-adasyn	0.519	0.342	0.208	0.188	0.647	0.047	0.325
flat-borderline	0.538	0.371	0.097	0.192	0.671	0.053	0.320
flat-ros	0.561	0.398	0.049	0.190	0.688	0.053	0.323
flat-smote	0.515	0.354	0.099	0.190	0.655	0.053	0.311
flat-smote-enn	0.384	0.268	0.081	0.185	0.446	0.027	0.232
flat-smote-tomek	0.509	0.347	0.068	0.191	0.637	0.050	0.300
local-adasyn	0.518	0.306	0.057	0.130	0.635	0.078	0.287
local-borderline	0.531	0.343	0.086	0.134	0.651	0.074	0.303
local-ros	0.558	0.350	0.144	0.150	0.685	0.087	0.329
local-smote	0.530	0.323	0.049	0.155	0.647	0.088	0.299
local-smote-enn	0.398	0.237	0.100	0.152	0.443	0.078	0.235
local-smote-tomek	0.524	0.320	0.107	0.155	0.638	0.086	0.305
local-selective	0.485	0.294	0.160	0.155	0.613	0.089	0.299

Table 57 – Mean macro-averaged F1 score for the LCPN experiments, for the RYDLS C-19 IDC dataset

Strategy	DT	kNN	MLP	NB	RF	SVM	Avg.
baseline	0.330	0.230	0.284	0.256	0.248	0.160	0.251
flat-adasyn	0.241	0.347	0.344	0.241	0.350	0.213	0.289
flat-borderline	0.262	0.309	0.351	0.234	0.298	0.258	0.285
flat-ros	0.257	0.355	0.315	0.212	0.308	0.231	0.280
flat-smote	0.281	0.338	0.335	0.267	0.356	0.226	0.301
flat-smote-enn	0.260	0.310	0.301	0.256	0.358	0.220	0.284
flat-smote-tomek	0.312	0.338	0.325	0.266	0.353	0.226	0.303
local-adasyn	0.254	0.308	0.250	0.268	0.378	0.192	0.275
local-borderline	0.250	0.316	0.348	0.288	0.346	0.213	0.294
local-ros	0.300	0.352	0.304	0.265	0.339	0.198	0.293
local-smote	0.264	0.323	0.310	0.278	0.374	0.198	0.291
local-smote-enn	0.273	0.258	0.310	0.269	0.339	0.179	0.271
local-smote-tomek	0.297	0.323	0.321	0.278	0.363	0.198	0.297
local-selective	0.332	0.312	0.238	0.269	0.335	0.177	0.277

Table 58 – Mean macro-averaged F1 score for the LCPN experiments, for the Diptera dataset

Strategy	DT	kNN	MLP	NB	RF	SVM	Avg.
baseline	0.395	0.441	0.493	0.301	0.505	0.473	0.435
flat-adasyn	0.390	0.418	0.459	0.277	0.490	0.424	0.410
flat-borderline	0.401	0.435	0.473	0.294	0.497	0.449	0.425
flat-ros	0.395	0.429	0.479	0.291	0.502	0.448	0.424
flat-smote	0.399	0.433	0.480	0.293	0.498	0.450	0.426
flat-smote-enn	0.412	0.414	0.440	0.309	0.456	0.420	0.409
flat-smote-tomek	0.408	0.434	0.467	0.300	0.498	0.451	0.426
local-adasyn	0.395	0.408	0.462	0.264	0.511	0.463	0.417
local-borderline	0.395	0.417	0.465	0.273	0.513	0.468	0.422
local-ros	0.400	0.417	0.478	0.286	0.510	0.478	0.428
local-smote	0.402	0.425	0.483	0.288	0.515	0.480	0.432
local-smote-enn	0.426	0.415	0.458	0.295	0.482	0.450	0.421
local-smote-tomek	0.407	0.430	0.481	0.290	0.512	0.479	0.433
local-selective	0.408	0.421	0.463	0.287	0.509	0.474	0.427

Table 59 – Mean macro-averaged F1 score for the LCPN experiments, for the ImCLEF07D dataset

Strategy	DT	kNN	MLP	NB	RF	SVM	Avg.
baseline	0.299	0.613	0.538	0.316	0.410	0.571	0.458
flat-adasyn	0.294	0.605	0.573	0.270	0.542	0.663	0.491
flat-borderline	0.308	0.621	0.574	0.264	0.526	0.663	0.493
flat-ros	0.326	0.639	0.549	0.267	0.526	0.670	0.496
flat-smote	0.320	0.610	0.572	0.294	0.560	0.666	0.504
flat-smote-enn	0.317	0.589	0.506	0.288	0.577	0.643	0.487
flat-smote-tomek	0.321	0.610	0.565	0.294	0.563	0.666	0.503
local-adasyn	0.304	0.560	0.558	0.266	0.500	0.658	0.474
local-borderline	0.316	0.559	0.581	0.248	0.509	0.657	0.478
local-ros	0.327	0.623	0.599	0.297	0.490	0.656	0.499
local-smote	0.311	0.589	0.572	0.315	0.516	0.664	0.495
local-smote-enn	0.318	0.540	0.556	0.299	0.536	0.618	0.478
local-smote-tomek	0.313	0.589	0.533	0.316	0.508	0.664	0.487
local-selective	0.300	0.575	0.581	0.321	0.489	0.649	0.486

Table 60 – Mean macro-averaged F1 score for the LCPN experiments, for the iCOPE dataset

Strategy	DT	kNN	MLP	NB	RF	SVM	Avg.
baseline	0.223	0.189	0.286	0.240	0.262	0.121	0.220
flat-adasyn	0.168	0.179	0.252	0.229	0.236	0.075	0.190
flat-borderline	0.162	0.181	0.231	0.243	0.264	0.175	0.209
flat-ros	0.296	0.189	0.221	0.262	0.274	0.188	0.238
flat-smote	0.191	0.118	0.202	0.223	0.306	0.150	0.198
flat-smote-enn	0.144	0.144	0.168	0.182	0.103	0.075	0.136
flat-smote-tomek	0.248	0.162	0.243	0.225	0.195	0.066	0.190
local-adasyn	0.265	0.248	0.307	0.226	0.296	0.073	0.236
local-borderline	0.278	0.219	0.252	0.231	0.212	0.243	0.239
local-ros	0.248	0.232	0.338	0.265	0.234	0.255	0.262
local-smote	0.273	0.224	0.281	0.202	0.289	0.223	0.249
local-smote-enn	0.230	0.168	0.185	0.217	0.239	0.075	0.186
local-smote-tomek	0.302	0.188	0.299	0.231	0.293	0.209	0.254
local-selective	0.225	0.226	0.271	0.242	0.226	0.224	0.236

Table 61 – Mean macro-averaged F1 score for the LCPN experiments, for the Instrument dataset

Strategy	DT	kNN	MLP	NB	RF	SVM	Avg.
baseline	0.535	0.620	0.006	0.275	0.674	0.258	0.395
flat-adasyn	0.529	0.746	0.039	0.277	0.770	0.503	0.477
flat-borderline	0.520	0.723	0.018	0.278	0.741	0.496	0.463
flat-ros	0.524	0.717	0.038	0.296	0.703	0.547	0.471
flat-smote	0.531	0.739	0.046	0.278	0.770	0.543	0.485
flat-smote-enn	0.519	0.706	0.008	0.277	0.752	0.531	0.466
flat-smote-tomek	0.529	0.738	0.054	0.278	0.764	0.542	0.484
local-adasyn	0.495	0.711	0.078	0.267	0.746	0.433	0.455
local-borderline	0.510	0.690	0.016	0.261	0.735	0.434	0.441
local-ros	0.502	0.695	0.035	0.279	0.702	0.494	0.451
local-smote	0.528	0.705	0.024	0.284	0.760	0.496	0.466
local-smote-enn	0.504	0.667	0.069	0.279	0.715	0.475	0.452
local-smote-tomek	0.520	0.705	0.023	0.284	0.745	0.495	0.462
local-selective	0.503	0.673	0.415	0.279	0.729	0.478	0.513

Table 62 – Mean Macro-Avg F1 Score for the LCN experiments, considering all datasets

Strategy	DT	kNN	MLP	NB	RF	SVM	Avg.
baseline	0.334	0.444	0.253	0.236	0.341	0.208	0.303
flat-adasyn	0.300	0.442	0.263	0.208	0.356	0.226	0.299
flat-borderline	0.306	0.446	0.258	0.205	0.364	0.242	0.304
flat-ros	0.304	0.500	0.257	0.215	0.349	0.239	0.311
flat-smote	0.307	0.433	0.261	0.220	0.366	0.234	0.304
flat-smote-enn	0.278	0.448	0.232	0.209	0.332	0.232	0.289
flat-smote-tomek	0.314	0.452	0.266	0.217	0.368	0.236	0.309
local-adasyn	0.317	0.471	0.359	0.211	0.428	0.314	0.350
local-borderline	0.323	0.484	0.348	0.208	0.412	0.320	0.349
local-ros	0.321	0.519	0.364	0.236	0.387	0.339	0.361
local-smote	0.342	0.490	0.371	0.232	0.439	0.346	0.370
local-smote-enn	0.340	0.415	0.339	0.243	0.421	0.326	0.347
local-smote-tomek	0.337	0.482	0.361	0.233	0.442	0.340	0.366
threshold-adasyn	0.328	0.476	0.351	0.219	0.426	0.336	0.356
threshold-borderline	0.324	0.486	0.339	0.221	0.406	0.334	0.352
threshold-ros	0.311	0.516	0.339	0.243	0.404	0.350	0.361
threshold-smote	0.336	0.497	0.321	0.233	0.450	0.350	0.365
threshold-smote-enn	0.335	0.430	0.343	0.244	0.439	0.350	0.357
threshold-smote-tomek	0.324	0.489	0.361	0.231	0.453	0.346	0.367
local-selective	0.306	0.431	0.369	0.245	0.414	0.342	0.351

Table 63 – Mean macro-averaged F1 score for the LCN experiments, for the Actinopterygii dataset

Strategy	DT	kNN	MLP	NB	RF	SVM	Avg.
baseline	0.420	0.299	0.055	0.161	0.500	0.043	0.246
flat-adasyn	0.403	0.305	0.057	0.172	0.482	0.044	0.244
flat-borderline	0.422	0.335	0.059	0.177	0.502	0.050	0.258
flat-ros	0.410	0.352	0.041	0.180	0.508	0.046	0.256
flat-smote	0.414	0.316	0.042	0.178	0.495	0.044	0.248
flat-smote-enn	0.353	0.289	0.050	0.160	0.396	0.050	0.216
flat-smote-tomek	0.405	0.314	0.060	0.177	0.480	0.045	0.247
local-adasyn	0.436	0.353	0.070	0.137	0.620	0.103	0.287
local-borderline	0.440	0.367	0.051	0.138	0.593	0.098	0.281
local-ros	0.416	0.388	0.054	0.154	0.552	0.106	0.278
local-smote	0.435	0.366	0.080	0.158	0.612	0.108	0.293
local-smote-enn	0.417	0.336	0.067	0.158	0.539	0.086	0.267
local-smote-tomek	0.433	0.368	0.061	0.159	0.608	0.110	0.290
threshold-adasyn	0.437	0.375	0.079	0.154	0.638	0.146	0.305
threshold-borderline	0.440	0.380	0.071	0.155	0.600	0.146	0.299
threshold-ros	0.414	0.401	0.080	0.159	0.555	0.148	0.293
threshold-smote	0.441	0.376	0.067	0.161	0.626	0.149	0.303
threshold-smote-enn	0.442	0.375	0.057	0.163	0.610	0.147	0.299
threshold-smote-tomek	0.437	0.375	0.090	0.162	0.629	0.149	0.307
local-selective	0.418	0.330	0.207	0.161	0.573	0.107	0.299

Table 64 – Mean macro-averaged F1 score for the LCN experiments, for the RYDLS C-19 IDC dataset

Strategy	DT	kNN	MLP	NB	RF	SVM	Avg.
baseline	0.412	0.627	0.088	0.216	0.259	0.197	0.300
flat-adasyn	0.318	0.392	0.140	0.171	0.286	0.133	0.240
flat-borderline	0.318	0.448	0.134	0.161	0.282	0.156	0.250
flat-ros	0.296	0.753	0.147	0.163	0.255	0.136	0.292
flat-smote	0.328	0.449	0.150	0.181	0.290	0.139	0.256
flat-smote-enn	0.288	0.559	0.140	0.171	0.285	0.139	0.264
flat-smote-tomek	0.337	0.479	0.158	0.189	0.299	0.126	0.265
local-adasyn	0.306	0.470	0.206	0.183	0.320	0.227	0.285
local-borderline	0.322	0.581	0.183	0.174	0.320	0.204	0.297
local-ros	0.329	0.737	0.207	0.169	0.265	0.214	0.320
local-smote	0.370	0.601	0.217	0.171	0.347	0.226	0.322
local-smote-enn	0.336	0.391	0.234	0.214	0.289	0.199	0.277
local-smote-tomek	0.357	0.577	0.218	0.167	0.359	0.199	0.313
threshold-adasyn	0.317	0.472	0.182	0.184	0.317	0.245	0.286
threshold-borderline	0.350	0.575	0.167	0.175	0.295	0.209	0.295
threshold-ros	0.310	0.737	0.173	0.177	0.316	0.211	0.321
threshold-smote	0.368	0.600	0.163	0.179	0.347	0.230	0.315
threshold-smote-enn	0.317	0.391	0.171	0.205	0.287	0.200	0.262
threshold-smote-tomek	0.373	0.574	0.153	0.174	0.377	0.200	0.309
local-selective	0.223	0.343	0.279	0.232	0.306	0.227	0.268

Table 65 – Mean macro-averaged F1 score for the LCN experiments, for the Diptera dataset

Strategy	DT	kNN	MLP	NB	RF	SVM	Avg.
baseline	0.312	0.397	0.381	0.296	0.395	0.312	0.349
flat-adasyn	0.310	0.387	0.361	0.272	0.376	0.294	0.333
flat-borderline	0.309	0.403	0.356	0.289	0.388	0.299	0.341
flat-ros	0.310	0.399	0.340	0.286	0.393	0.302	0.338
flat-smote	0.312	0.401	0.385	0.286	0.387	0.303	0.346
flat-smote-enn	0.337	0.422	0.366	0.290	0.403	0.373	0.365
flat-smote-tomek	0.318	0.408	0.372	0.291	0.393	0.316	0.350
local-adasyn	0.326	0.400	0.472	0.251	0.491	0.467	0.401
local-borderline	0.323	0.400	0.463	0.264	0.484	0.465	0.400
local-ros	0.311	0.407	0.482	0.295	0.437	0.484	0.403
local-smote	0.331	0.416	0.490	0.304	0.496	0.490	0.421
local-smote-enn	0.332	0.413	0.480	0.302	0.488	0.484	0.417
local-smote-tomek	0.330	0.417	0.480	0.305	0.493	0.490	0.419
threshold-adasyn	0.321	0.405	0.476	0.274	0.492	0.485	0.409
threshold-borderline	0.319	0.405	0.463	0.286	0.487	0.481	0.407
threshold-ros	0.314	0.409	0.491	0.297	0.440	0.488	0.407
threshold-smote	0.333	0.418	0.486	0.305	0.495	0.492	0.422
threshold-smote-enn	0.336	0.419	0.487	0.304	0.489	0.492	0.421
threshold-smote-tomek	0.332	0.419	0.491	0.304	0.493	0.491	0.422
local-selective	0.325	0.414	0.482	0.305	0.486	0.489	0.417

Table 66 – Mean macro-averaged F1 score for the LCN experiments, for the ImCLEF07D dataset

Strategy	DT	kNN	MLP	NB	RF	SVM	Avg.
baseline	0.223	0.538	0.466	0.287	0.274	0.446	0.372
flat-adasyn	0.196	0.589	0.469	0.200	0.283	0.510	0.375
flat-borderline	0.195	0.596	0.473	0.188	0.269	0.508	0.372
flat-ros	0.206	0.616	0.482	0.204	0.298	0.517	0.387
flat-smote	0.213	0.587	0.469	0.219	0.317	0.510	0.386
flat-smote-enn	0.227	0.576	0.470	0.219	0.317	0.503	0.385
flat-smote-tomek	0.213	0.590	0.470	0.218	0.315	0.508	0.386
local-adasyn	0.202	0.597	0.471	0.231	0.297	0.551	0.392
local-borderline	0.206	0.607	0.467	0.234	0.281	0.545	0.390
local-ros	0.217	0.628	0.482	0.278	0.336	0.561	0.417
local-smote	0.218	0.619	0.485	0.287	0.334	0.549	0.415
local-smote-enn	0.234	0.534	0.492	0.282	0.359	0.559	0.410
local-smote-tomek	0.222	0.619	0.473	0.286	0.329	0.557	0.414
threshold-adasyn	0.216	0.616	0.472	0.227	0.301	0.564	0.399
threshold-borderline	0.214	0.627	0.481	0.223	0.293	0.554	0.399
threshold-ros	0.223	0.631	0.475	0.284	0.340	0.568	0.420
threshold-smote	0.218	0.635	0.477	0.287	0.343	0.556	0.419
threshold-smote-enn	0.230	0.558	0.505	0.285	0.383	0.577	0.423
threshold-smote-tomek	0.220	0.634	0.481	0.286	0.336	0.554	0.419
local-selective	0.207	0.582	0.453	0.283	0.323	0.527	0.396

Table 67 – Mean macro-averaged F1 score for the LCN experiments, for the iCOPE dataset

Strategy	DT	kNN	MLP	NB	RF	SVM	Avg.
baseline	0.245	0.220	0.256	0.239	0.223	0.146	0.222
flat-adasyn	0.193	0.237	0.304	0.237	0.270	0.174	0.236
flat-borderline	0.210	0.174	0.262	0.222	0.298	0.215	0.230
flat-ros	0.228	0.164	0.257	0.245	0.219	0.203	0.219
flat-smote	0.183	0.111	0.304	0.262	0.243	0.155	0.210
flat-smote-enn	0.069	0.137	0.139	0.212	0.143	0.075	0.129
flat-smote-tomek	0.208	0.187	0.254	0.228	0.251	0.175	0.217
local-adasyn	0.240	0.264	0.317	0.242	0.285	0.119	0.245
local-borderline	0.238	0.222	0.304	0.196	0.238	0.184	0.230
local-ros	0.264	0.227	0.312	0.254	0.221	0.178	0.243
local-smote	0.268	0.198	0.307	0.217	0.249	0.219	0.243
local-smote-enn	0.296	0.170	0.212	0.253	0.255	0.160	0.224
local-smote-tomek	0.249	0.170	0.293	0.228	0.273	0.201	0.236
threshold-adasyn	0.278	0.237	0.260	0.261	0.243	0.130	0.235
threshold-borderline	0.211	0.196	0.296	0.260	0.212	0.171	0.224
threshold-ros	0.195	0.188	0.344	0.272	0.270	0.179	0.241
threshold-smote	0.221	0.212	0.264	0.210	0.289	0.170	0.228
threshold-smote-enn	0.241	0.193	0.314	0.250	0.265	0.200	0.244
threshold-smote-tomek	0.149	0.191	0.317	0.208	0.282	0.186	0.222
local-selective	0.257	0.220	0.289	0.242	0.226	0.237	0.245

Table 68 – Mean macro-averaged F1 score for the LCN experiments, for the Instrument dataset

Strategy	DT	kNN	MLP	NB	RF	SVM	Avg.
baseline	0.390	0.583	0.274	0.218	0.397	0.106	0.328
flat-adasyn	0.383	0.740	0.248	0.196	0.439	0.200	0.368
flat-borderline	0.385	0.722	0.265	0.195	0.445	0.222	0.372
flat-ros	0.372	0.714	0.274	0.212	0.420	0.230	0.370
flat-smote	0.390	0.736	0.216	0.196	0.462	0.252	0.375
flat-smote-enn	0.391	0.705	0.229	0.202	0.451	0.251	0.372
flat-smote-tomek	0.402	0.735	0.281	0.199	0.468	0.246	0.389
local-adasyn	0.395	0.742	0.622	0.221	0.554	0.419	0.492
local-borderline	0.407	0.726	0.622	0.243	0.558	0.421	0.496
local-ros	0.387	0.727	0.646	0.264	0.511	0.489	0.504
local-smote	0.428	0.740	0.649	0.253	0.593	0.483	0.524
local-smote-enn	0.424	0.644	0.551	0.250	0.597	0.469	0.489
local-smote-tomek	0.432	0.740	0.644	0.251	0.592	0.484	0.524
threshold-adasyn	0.396	0.749	0.638	0.211	0.565	0.444	0.501
threshold-borderline	0.413	0.730	0.555	0.229	0.547	0.446	0.487
threshold-ros	0.408	0.729	0.468	0.268	0.504	0.503	0.480
threshold-smote	0.434	0.741	0.469	0.255	0.598	0.501	0.500
threshold-smote-enn	0.442	0.646	0.526	0.255	0.601	0.487	0.493
threshold-smote-tomek	0.435	0.741	0.636	0.255	0.600	0.498	0.528
local-selective	0.408	0.699	0.506	0.248	0.572	0.464	0.483

Table 69 – Difference between the baseline and each resampling strategy for the LCPN scenario, for each dataset

Strategy	Actinopterygii	RYDLS C-19 IDC	Diptera	ImCLEF07D	iCOPE	Instrument
baseline	0.317	0.251	0.435	0.458	0.220	0.395
flat-adasyn	0.008	0.038	-0.025	0.033	-0.030	0.083
flat-borderline	0.003	0.034	-0.010	0.035	-0.011	0.068
flat-ros	0.006	0.028	-0.011	0.038	0.018	0.076
flat-smote	-0.006	0.049	-0.009	0.046	-0.022	0.090
flat-smote-enn	-0.086	0.033	-0.026	0.029	-0.084	0.071
flat-smote-tomek	-0.017	0.052	-0.008	0.045	-0.030	0.090
local-adasyn	-0.030	0.024	-0.018	0.017	0.016	0.060
local-borderline	-0.014	0.042	-0.013	0.021	0.019	0.046
local-ros	0.012	0.042	-0.007	0.041	0.042	0.057
local-selective	-0.019	0.040	-0.003	0.037	0.029	0.072
local-smote	-0.083	0.020	-0.014	0.020	-0.035	0.057
local-smote-enn	-0.012	0.045	-0.002	0.029	0.034	0.067
local-smote-tomek	-0.018	0.026	-0.008	0.028	0.016	0.118

Table 70 – Percentage variation between the baseline and each resampling strategy for the LCPN scenario, for each dataset

Strategy	Actinopterygii (%)	RYDLS C-19 IDC(%)	Diptera (%)	ImCLEF07D (%)	iCOPE (%)	Instrument (%)
flat-adasyn	2.47	15.12	-5.75	7.28	-13.78	20.95
flat-borderline	0.95	13.53	-2.26	7.61	-4.92	17.23
flat-ros	1.84	11.27	-2.45	8.37	8.25	19.30
flat-smote	-2.00	19.56	-2.11	10.01	-9.92	22.76
flat-smote-enn	-26.94	13.06	-6.02	6.30	-38.23	17.95
flat-smote-tomek	-5.36	20.69	-1.92	9.90	-13.78	22.68
local-adasyn	-9.45	9.42	-4.03	3.60	7.12	15.29
local-borderline	-4.46	16.78	-2.95	4.48	8.63	11.74
local-ros	3.68	16.58	-1.50	8.92	19.00	14.32
local-selective	-5.88	15.85	-0.58	8.01	12.94	18.12
local-smote	-26.05	7.96	-3.14	4.37	-15.67	14.40
local-smote-enn	-3.89	18.04	-0.35	6.41	15.22	17.06
local-smote-tomek	-5.67	10.28	-1.76	6.12	7.04	29.94

Table 71 – Difference between the baseline and each resampling strategy for the LCN scenario, for each dataset

Strategy	Actinopterygii	RYDLS C-19 IDC	Diptera	ImCLEF07D	iCOPE	Instrument
baseline	0.317	0.251	0.435	0.458	0.220	0.395
flat-adasyn	-0.002	-0.060	-0.016	0.002	0.014	0.040
flat-borderline	0.011	-0.050	-0.008	-0.001	0.009	0.044
flat-ros	0.010	-0.008	-0.011	0.015	-0.002	0.042
flat-smote	0.002	-0.044	-0.003	0.014	-0.012	0.047
flat-smote-enn	-0.030	-0.036	0.016	0.013	-0.092	0.044
flat-smote-tomek	0.001	-0.035	0.001	0.013	-0.004	0.061
local-adasyn	0.040	-0.015	0.052	0.019	0.023	0.164
local-borderline	0.035	-0.003	0.051	0.018	0.009	0.168
local-ros	0.032	0.020	0.054	0.045	0.021	0.176
local-smote	0.047	0.022	0.072	0.043	0.022	0.196
local-smote-enn	0.021	-0.023	0.068	0.038	0.003	0.161
local-smote-tomek	0.044	0.013	0.070	0.042	0.014	0.196
threshold-adasyn	0.059	-0.014	0.060	0.027	0.013	0.173
threshold-borderline	0.052	-0.005	0.058	0.026	0.003	0.159
threshold-ros	0.047	0.021	0.058	0.048	0.020	0.152
threshold-smote	0.057	0.015	0.073	0.047	0.006	0.172
threshold-smote-enn	0.053	-0.038	0.072	0.051	0.022	0.165
threshold-smote-tomek	0.061	0.009	0.073	0.046	0.001	0.200
local-selective	0.053	-0.032	0.068	0.024	0.024	0.155

Table 72 – Percentage variation between the baseline and each resampling strategy for the LCN scenario, for each dataset

Strategy	Actinopterygii (%)	RYDLS C-19 IDC (%)	Diptera (%)	ImCLEF07D (%)	iCOPE (%)	Instrument (%)
flat-adasyn	-0.79	-23.81	-3.57	0.47	6.51	10.05
flat-borderline	3.52	-19.89	-1.88	-0.18	3.94	11.23
flat-ros	3.10	-3.25	-2.42	3.24	-0.98	10.73
flat-smote	0.58	-17.37	-0.73	2.95	-5.37	11.99
flat-smote-enn	-9.45	-14.39	3.76	2.84	-41.94	11.02
flat-smote-tomek	0.16	-13.99	0.19	2.91	-1.97	15.33
local-adasyn	12.66	-5.77	12.04	4.19	10.45	41.60
local-borderline	10.98	-0.99	11.73	3.86	4.01	42.61
local-ros	10.08	8.09	12.38	9.76	9.61	44.59
local-smote	14.76	8.82	16.64	9.39	9.77	49.75
local-smote-enn	6.57	-9.02	15.57	8.23	1.29	40.84
local-smote-tomek	13.71	5.17	16.18	9.17	6.43	49.62
threshold-adasyn	18.43	-5.44	13.80	5.90	6.06	43.71
threshold-borderline	16.49	-1.86	13.34	5.75	1.29	40.20
threshold-ros	14.65	8.29	13.27	10.45	9.01	38.51
threshold-smote	17.96	5.84	16.72	10.27	2.80	43.50
threshold-smote-enn	16.60	-15.12	16.64	11.07	10.14	41.77
threshold-smote-tomek	19.12	3.45	16.76	10.08	0.30	50.55
local-selective	16.70	-12.53	15.64	5.13	10.75	39.23
local-selective	0.053	-0.032	0.068	0.024	0.024	0.155

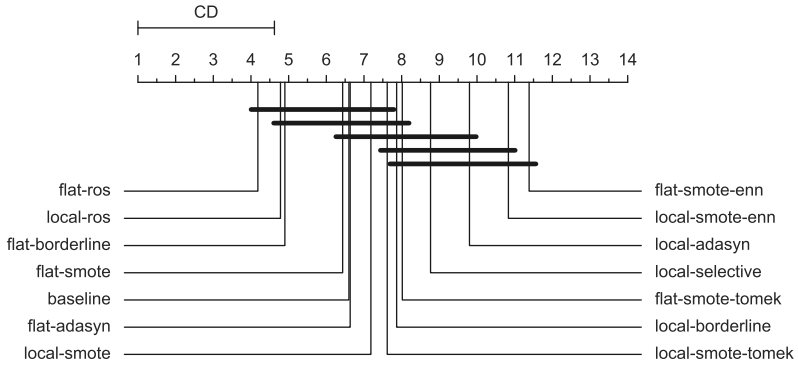


Figure 34 – CD plot from Nemenyi's test ($\alpha = 0.05$) for the Actinopterygii dataset, comparing the resampling strategies used with the LCPN Hierarchical classifier

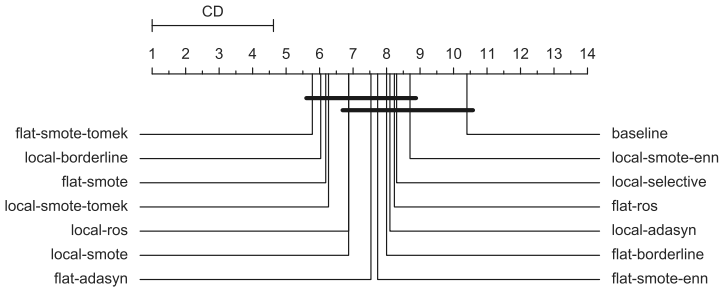


Figure 35 – CD plot from Nemenyi's test ($\alpha = 0.05$) for the RYDLS C-19 IDC dataset, comparing the resampling strategies used with the LCPN Hierarchical classifier

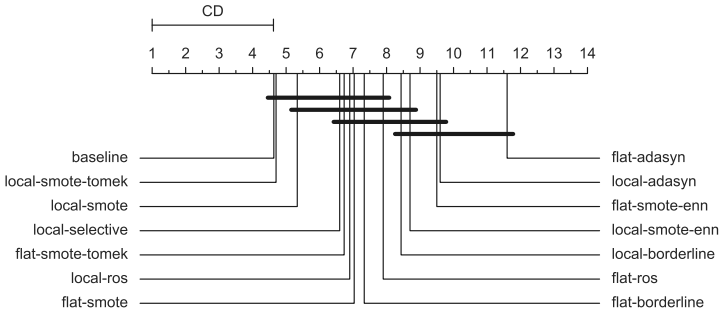


Figure 36 – CD plot from Nemenyi's test ($\alpha = 0.05$) for the Diptera dataset, comparing the resampling strategies used with the LCPN Hierarchical classifier

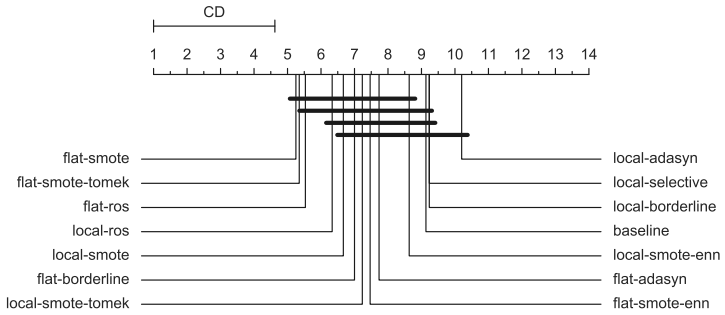


Figure 37 – CD plot from Nemenyi's test ($\alpha = 0.05$) for the ImCLEF07D dataset, comparing the resampling strategies used with the LCPN Hierarchical classifier

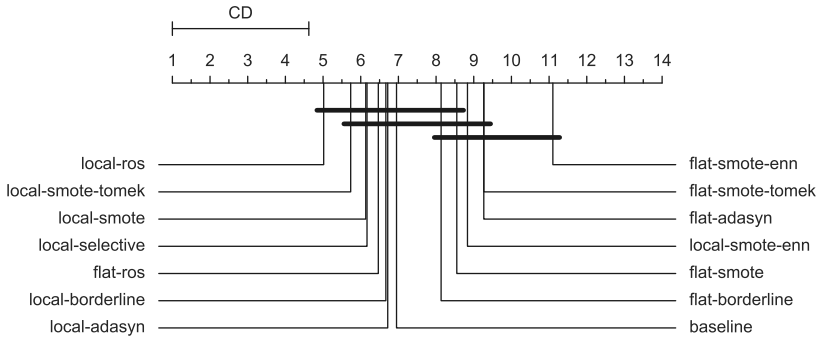


Figure 38 – CD plot from Nemenyi's test ($\alpha = 0.05$) for the iCOPE dataset, comparing the resampling strategies used with the LCPN Hierarchical classifier

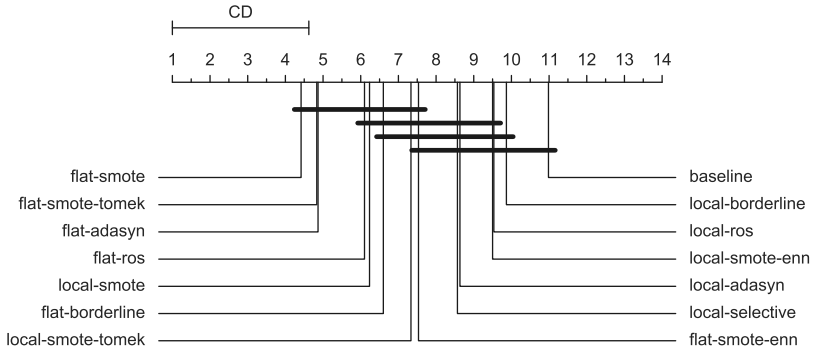


Figure 39 – CD plot from Nemenyi's test ($\alpha = 0.05$) for the Instrument dataset, comparing the resampling strategies used with the LCPN Hierarchical classifier

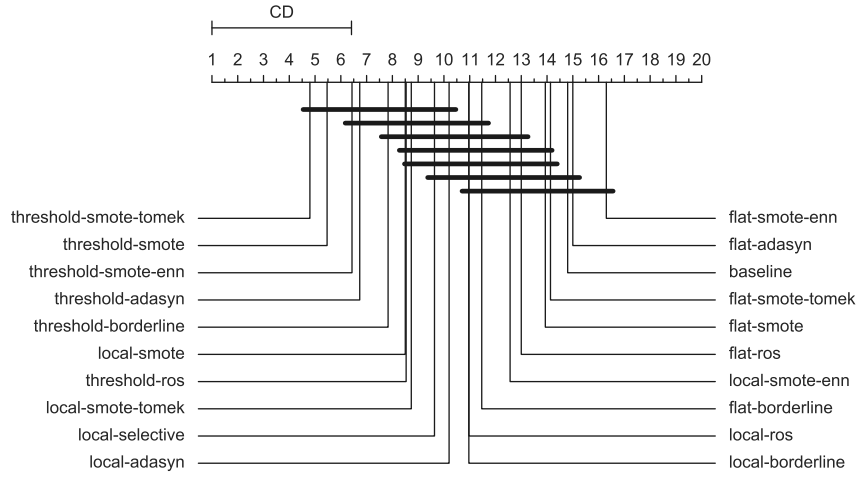


Figure 40 – CD plot from Nemenyi's test ($\alpha = 0.05$) for the Actinopterygii dataset, comparing the resampling strategies used with the LCN Hierarchical classifier

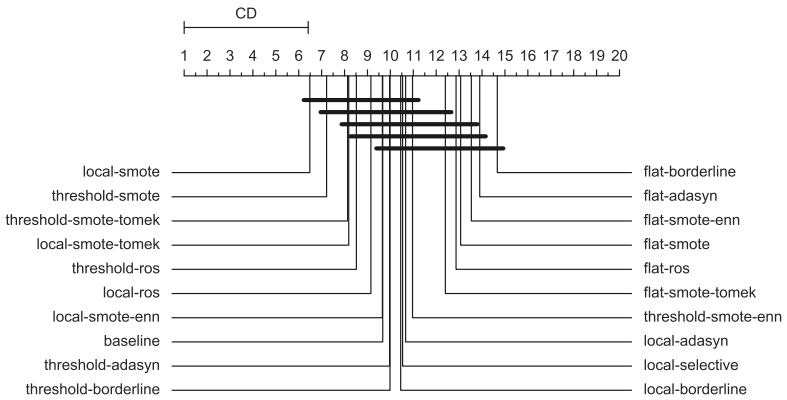


Figure 41 – CD plot from Nemenyi's test ($\alpha = 0.05$) for the RYDLS C-19 IDC dataset, comparing the resampling strategies used with the LCN Hierarchical classifier

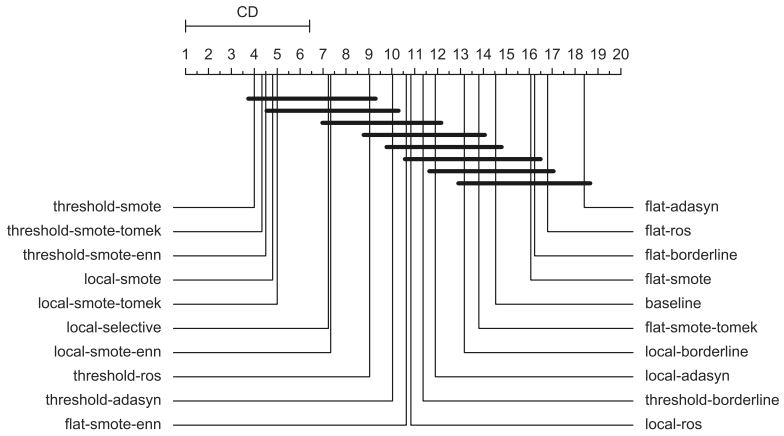


Figure 42 – CD plot from Nemenyi's test ($\alpha = 0.05$) for the Diptera dataset, comparing the resampling strategies used with the LCN Hierarchical classifier

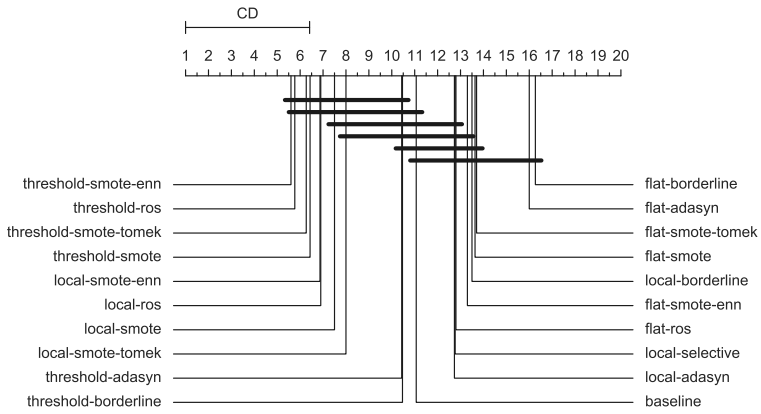


Figure 43 – CD plot from Nemenyi's test ($\alpha = 0.05$) for the ImCLEF07D dataset, comparing the resampling strategies used with the LCN Hierarchical classifier

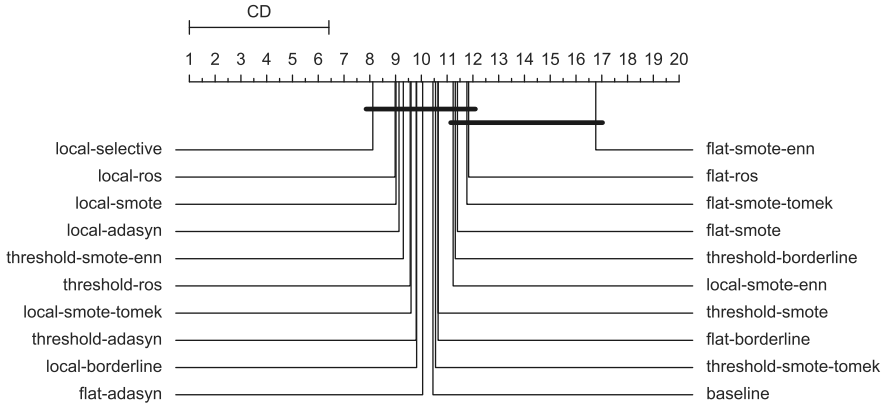


Figure 44 – CD plot from Nemenyi's test ($\alpha = 0.05$) for the iCOPE dataset, comparing the resampling strategies used with the LCN Hierarchical classifier

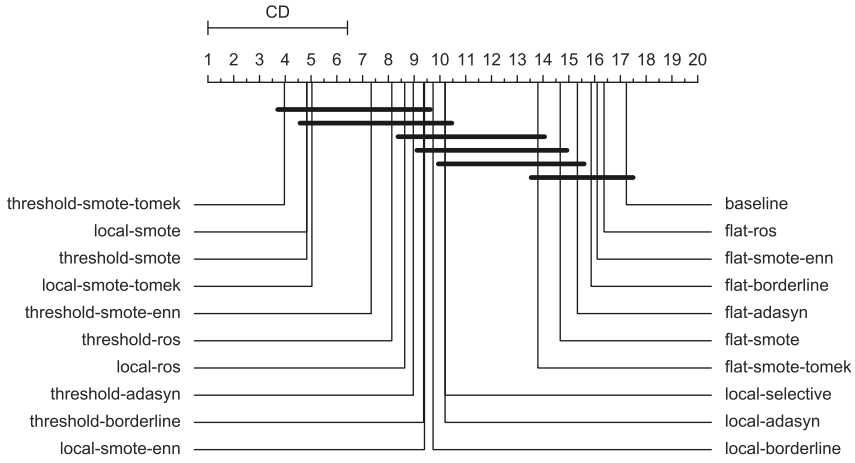


Figure 45 – CD plot from Nemenyi's test ($\alpha = 0.05$) for the Instrument dataset, comparing the resampling strategies used with the LCN Hierarchical classifier