Automatic Disease Vector Mosquitoes Identification via Hierarchical Data Stream Classification*

Eduardo Tieppo
Pontifícia Universidade Católica do Paraná (PUCPR) - PPGIa
Curitiba, Brazil
eduardo.tieppo@ppgia.pucpr.br

Jean Paul Barddal
Pontifícia Universidade Católica do Paraná (PUCPR) - PPGIa
Curitiba, Brazil
jean.barddal@ppgia.pucpr.br

Júlio Cesar Nievola
Pontifícia Universidade Católica do Paraná (PUCPR) - PPGIa
Curitiba, Brazil
nievola@ppgia.pucpr.br

ABSTRACT

Vector-borne diseases (VBDs), such as Dengue or Malaria, are one of the main concerns of public health agencies and governments. These diseases are mainly spread by mosquitoes acting as vectors by transmitting infected blood between humans. Machine learning can be used to design and improve control strategies of VBDs by providing models able to recognize disease vector mosquitoes and automatically capture or kill harmful species. The automatic identification of disease vector mosquitoes was not yet addressed concerning the hierarchical classification of data streams. Thus, reliable information has not been used to improve learning models, such as mosquitoes' hierarchical taxonomy. In this study, we propose a framework for the automatic identification of disease vector mosquitoes in the context of the hierarchical classification of data streams area. To this end, we propose a hierarchical adaptation of a disease vector mosquitoes' dataset to include their taxonomy and introduce kNC and Dribble, two novel classification methods fitted to hierarchical data streams representing the mosquitoes. Results depicted that our framework, using summarization techniques, achieves significantly better prediction and processing speed rates when compared to existing state-of-the-art models.

CCS CONCEPTS

• **Applied computing** → **Bioinformatics**; • **Information systems** → **Data stream mining**; • **Computing methodologies** → **Supervised learning by classification**;

KEYWORDS

Vector-borne diseases, Hierarchical Classification, Data Stream Classification, Data Summarization

ACM Reference Format:
Eduardo Tieppo, Jean Paul Barddal, and Júlio Cesar Nievola. 2022. Automatic Disease Vector Mosquitoes Identification via Hierarchical Data Stream Classification. In *Proceedings of ACM SAC Conference (SAC'22).* ACM, New York, NY, USA, Article 4, 8 pages. https://doi.org/xx.xxx/xxx_x

*Supported by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

1 INTRODUCTION AND BACKGROUND

One of the most significant current challenges facing public health is the prevention and control of Vector-borne diseases (VBDs), which are diseases caused by living organisms, named vectors, that can transmit infectious diseases between humans [21].

According to the World Health Organization, VBDs, such as Dengue, yellow fever, chikungunya, Zika virus, and others, are responsible for more than one million deaths every year around the world [21].

Mosquitoes are the best-known disease vector. For instance, the *Aedes aegypti* mosquito can transmit Dengue, yellow fever, chikungunya, and Zika virus by carrying infected blood from one organism to another during a meal [17].

Diverse public health interventions have been carried out to reduce the spread of these disease vector mosquitoes and consequently prevent possible disease outbreaks. The identification of the mosquitoes is an essential step in the designing of vector control strategies [5].

However, in several cases, public health technicians perform this identification manually, which is not scalable without the commitment of many qualified specialists in a time-consuming task [7, 21]. Consequently, machine learning methods have been proposed for the automatic identification of disease vector mosquitoes or in the forecasting of outbreaks of VBDs [7, 14].

In general, machine learning techniques benefit from their ability in automating decision-making processes related to the control of VBDs by, for instance, classifying the species and sex of mosquitoes or capturing target mosquitoes using automatic traps with sensors [7].

Still, a drawback regards the amount of data available for the learning model, specially labeled data that allows the learning model to associate the recognized patterns to the target outputs.

In this sense, the authors in [25] introduced a new dataset obtained from mosquitoes' traps using optical sensors comprising nearly one million instances representing mosquito feature sets belonging to 17 distinct species, including the ones of disease vector mosquitoes, such as *Aedes aegypti* (Dengue, yellow fever, chikungunya, Zika virus), *Aedes albopictus* (Chikungunya, Dengue, West Nile virus) and *Culex quinquefasciatus* (Lymphatic filariasis) [21].

This dataset was proposed in the machine learning context as a real-world stream dataset fitted to the data stream classification task. Classification models often assume datasets to be static and entirely available for a well-defined training step. However, these assumptions no longer reflect many real-world scenarios, such as the automatic identification of disease vector mosquitoes. Data collection from sensors, for example, results in huge amounts of data being generated and processed over time.

Thus, data stream classification has been revisiting many aspects related to the usual classification task in the last years [10], requiring novel approaches that can be updated over time, use constrained computational resources, and are able to detect and adapt to changes in the data behavior, a phenomenon named *concept drift* [28].

In the automatic identification of disease vector mosquitoes, for example, changes in the environmental conditions, such as temperature, humidity, or air pressure, can alter the flying behavior of the mosquitoes, resulting in concept drifts in the collected data [25]. The dataset proposed in [25], hereafter referred to as "Insects dataset", was introduced as a flat dataset, even though the classes associated with the instances represent species of disease vector mosquitoes, which are naturally organized in an entomological taxonomy.

In contrast to the aforementioned flat classification, in hierarchical classification, instances are not associated with an independent label (class) but with a label path, where inner labels represent hierarchical relationships with the outer labels, accurately as in the taxonomic classification of species [24, 29].

In the work [24], the authors presented a comprehensive analysis of how hierarchical classification models previously reported in the literature performed when compared to non-hierarchical (flat) approaches, considering datasets of classification problems with pre-existing class hierarchies. As a result, more than 90% of the studies reported a "better prediction performance" when using hierarchical approaches.

Here, we are particularly interested in the intersection of hierarchical and data stream classification, instantiated by the automatic identification of disease vector mosquitoes. Several challenges arise from this intersection, as the constraints of both classification areas must be concomitantly accounted for. In other words, our goal is to bring forward machine learning models that are accurate when a class hierarchy is available, are updatable as new training data becomes available, and are computationally light-weighted in terms of processing time and memory consumption [4, 10].

The authors in [22] presented a similar approach by introducing a method for the classification of hierarchical data streams applied in an entomology-related dataset. However, the method is based on computationally intensive distance computations and, thus, is not suitable for real-world stream datasets, as the Insects dataset described above.

In this paper, we propose a framework for the automatic identification of disease vector mosquitoes in the context of the Hierarchical classification of data streams task. To this end, we adapt a dataset of disease vector mosquitoes to include the mosquitoes' entomological taxonomy and propose two novel methods for the classification of the hierarchical data streams representing the mosquitoes.

More specifically, first, we adapt the disease vector mosquitoes dataset proposed in [25] to include hierarchical label paths describing the taxonomy of mosquitoes. Then, we propose a framework for classifying disease vector mosquitoes in the Hierarchical classification of data streams context. To that, we adapt and apply summarization techniques, i.e., incremental centroids [26], and cluster feature vectors [30], as part of a hierarchical data stream classification process, and, at the core of the framework, we propose k-Nearest Centroids and Dribble, two novel methods for the hierarchical classification of data streams.

The remainder of this paper is organized as follows. Section 2 describes the problem of disease vector mosquitoes from the perspective of the hierarchical classification of data streams, while Section 3 brings forward related works. Section 4 describes the proposed framework for the automatic identification of disease vector mosquitoes in the context of the Hierarchical classification of data streams. Section 5 comprises the experiments used to test the framework and the comparisons performed with related works. Finally, Section 6 concludes this paper and states envisioned future works.

2 PROBLEM STATEMENT

In this section, we describe the problem of disease vector mosquitoes as an instance of a problem regarding the hierarchical classification of data streams.

Hierarchical data stream classification lies at the intersection of hierarchical classification and data streams. Therefore, it differs from traditional classification in two fundamental aspects. First, regarding hierarchical classification, instances (representing mosquitoes) are associated with not one independent label (class) but with a label path that belongs to a hierarchically structured set of classes representing the taxonomy of the mosquitoes. Second, regarding data stream classification, the entire dataset comprising instances for training is not available; instead, the instances (representing the mosquitoes) are presented sequentially, one by one, over time to the model [27].

More formally, we denote $hDS = [(\vec{x}^t, \vec{y}^t)]_{t=0}^{\infty}$ to be a hierarchical data stream providing instances $(\vec{x}^t, \vec{y}^t)$, each of which arriving at a timestamp t, where $\vec{x}^t$ is a d-dimensional features set and its values, and $\vec{y}^t$ is the corresponding ground-truth label path (hierarchically structured classes) for the given instance $\vec{x}^t$ [27].

The class labels are organized under a regular concept hierarchy that is structured on a partially ordered set $(Y, \succ)$, where Y is a finite set containing all concepts representing the problem and the relationship $\succ$ is defined as an asymmetric, anti-reflexive, and transitive subsumption relation [24].

Next, the hierarchical classification of data streams is formalized as $f^t : \vec{x}^t \mapsto \vec{y}^t$, such that an hypothesis f^t is continuously updated by mapping features $\vec{x}$ to the corresponding labels y^t accurately [10, 27]. Thus, regarding the problem of disease vector mosquitoes, a learning model $f^t : \vec{x}^t \mapsto \vec{y}^t$ receives the mapping features $\vec{x}$ representing the mosquitoes' features (e.g, wing-beat frequency) and predicts $\vec{y}^t$, i.e., the mosquitoes' taxonomy from the highest to the lowest level of the taxonomic hierarchy.

Besides, the time component is inherent in data streams, and thus, data streams are expected to be ephemeral, i.e., the underlying data distribution is expected to change, resulting in concept drifts [12, 28].

A concept (C) is defined as set of prior probabilities of the classes and class-conditional probability density function given by $C = \bigcup_{y \in Y}\{(P[y], P[\vec{x}|y])\}$ [3]. A concept drift occurs if, at between two timestamps t_i and $t_j = t_i + \Delta$ with $\Delta > 1$, $C^{t_i} \neq C^{t_j}$ holds [3, 28]. Consequently, f^t should capture the data dynamics accordingly.

As stated before, in the automatic identification of disease vector mosquitoes, environmental conditions can affect the wing-beat frequency of the mosquitoes and mislead stationary learning models [25].

Also related to the time component of data streams, algorithms must work within bounded computational resources, analyzing each instance only once according to their arrival [4]. The processing time of an arriving instance must not surpass the ratio in which new instances become available. Otherwise, the model will need to discard new instances or it will not adapt swiftly enough to handle concept drifts [3].

As mosquitoes' traps selectively capture species of disease vector mosquitoes and free other harmless species, any instability of the learning model could result in not capturing harmful species [25].

3 RELATED WORK

In this section, we focus on studies concerning the hierarchical classification of data streams. Note that studies related exclusively to the automatic classification of disease vector mosquitoes but regarding other research areas are outside the scope of this study. Despite effort spent in these related areas, several studies do not match problem constraints from the hierarchical classification of data streams area (cf. described in Section 2), by, for example, using stationary or small-sized datasets.

The literature has addressed hierarchical and data stream classification tasks individually, thus resulting in several works and comprehensive reviews of such studies. For instance, reviews on hierarchical classification [16, 24] and data stream classification [4, 11, 23] depict the recentness of approaches in both areas and indicate the lack of works lying at their intersection.

Recently, specific works focused on this intersection; however, the proposals do not consider all the constraints required from both fields simultaneously. For instance, the authors in [13] proposed a method that received a data stream as input but processed it in batches and overlooked concept drifts. Also, the authors in [6] proposed a hierarchical model learning scheme for non-hierarchical data streams [27].

Finally, in contrast to the aforementioned works, we highlight the instance-based hierarchical data stream classifier proposed in [22]. More specifically, the authors proposed an incremental method based on the traditional k-Nearest Neighbors (kNN) technique [2] representing the data hierarchically and using a memory buffer on nodes of the hierarchy to forget instances.

Nonetheless, this proposal has drawbacks w.r.t. data streaming requirements as the k-Nearest Neighbors technique requires computationally intensive distance computations. Therefore, it is of interest to adapt this method to summarize data adequately so that the number of distance computations is lessened [19, 26].

In this sense, in the non-hierarchical On-Demand classification method proposed in [1], the model stores representations of the data (statistical summaries) and performs the comparison of a new instance with the statistical summaries instead of the instances themselves, reducing the required number of comparisons and maintaining constant the memory used to store samples. These statistics summaries are, in fact, cluster feature vectors (*CFs*), described first in [30].

Similar proposals can be found in the data stream clustering algorithms StreamLS, and Stream k-Means [20], where the data stream is processed in chunks using centroids in each chunk, and in the CluStream method, which uses cluster feature vectors to construct actual clusters in an offline step [4].

4 THE PROPOSED FRAMEWORK

In this section, we propose a framework for the automatic identification of disease vector mosquitoes as a hierarchical classification of data streams task. This framework comprises k-Nearest Centroids (kNC) and Dribble, two novel methods for the classification of hierarchical data streams.

Figure 1 illustrates the main steps comprising the proposed framework. First, we process data to include the hierarchical taxonomy of mosquitoes and represent it in a data structure. Then, we apply a summarization technique to store/obtain a summary description of data. After that, models predict the mosquitoes' taxonomy from the highest to the lowest level of the taxonomic hierarchy using a top-down approach across the hierarchy. Finally, the model may update itself using incoming instances and forget older data to adapt to changes in data distribution. This process is repeated in a cycle for each incoming instance. All the above steps are detailed below.

The Insects dataset was introduced in [25] as a flat data stream. In this study, we incorporate the hierarchy naturally present in the classes of the dataset, i.e., the biological taxonomy of the mosquitoes. Figure 2 illustrates the resulting class hierarchy from the first common taxonomic rank (class Insecta, omitted) to the mosquitoes' species. Among the 17 species represented, the six marked with an asterisk represent species with sex distinction in the dataset, thus resulting in 24 hierarchical labels.

Note that even though some instances do not have all taxonomic levels described (n/a), it does not result in a partial depth problem

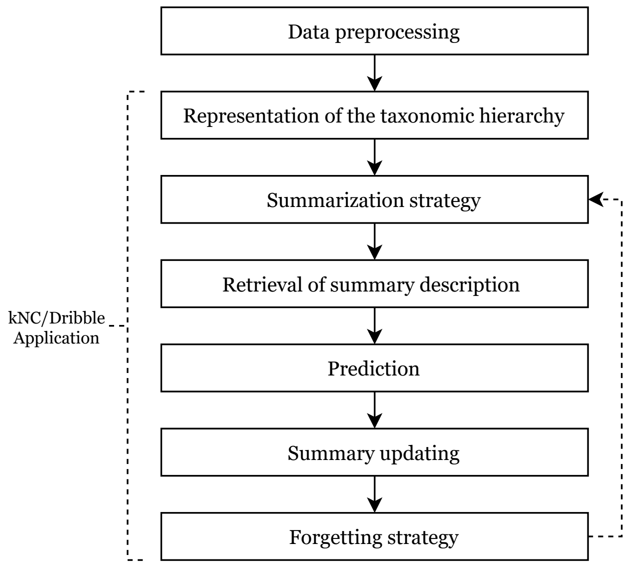

Figure 1: General view of the proposed framework.

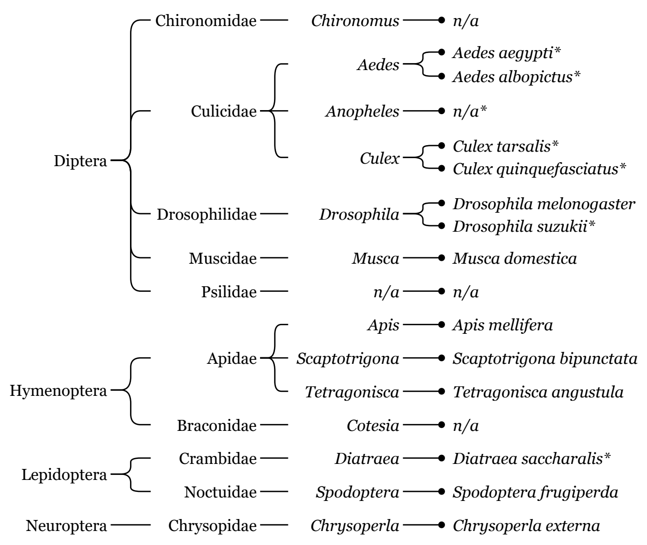

Figure 2: Representation of the class hierarchy on Insects dataset.

[24]. The lack of specific labels only denotes a failure in the data annotation process since the insects represented by the instances have a known taxonomy not described in the dataset.

Table 1 details the 11 hierarchically labeled datasets resulting from our hierarchical adaptation of the Insects Datasets proposed in [25]. Note that the "Insects-o-o-c" represents the main dataset, while other ones represent datasets built with different sampling strategies to simulate natural effects in insects' behavior.

Our hierarchical adaptation of the Insects dataset and its specification are freely available for download[1].

Besides the data preprocessing step, kNC or Dribble perform the other steps in the proposed framework. Both methods are interchangeable across the framework, and their suitability depends on the specific characteristics of a dataset/problem.

First, both methods organize class labels in a hierarchically structured class taxonomy using a tree data structure, with the paths from the root node to the leaf nodes representing label paths (classes) of instances/mosquitoes.

[1]http://www.ppgia.pucpr.br/~jean.barddal/datasets/local_knc_dribble.zip

Table 1: Hierarchically-labeled Insects dataset variants.

Dataset	Instances	Features	Classes	Labels per level
Insects-a-b	52,848	33	6	1, 1, 2, 6
Insects-a-i	355,275	33	6	1, 1, 2, 6
Insects-i-a-r-b	79,986	33	6	1, 1, 2, 6
Insects-i-a-r-i	452,044	33	6	1, 1, 2, 6
Insects-i-b	57,018	33	6	1, 1, 2, 6
Insects-i-g-b	24,15	33	6	1, 1, 2, 6
Insects-i-g-i	143,323	33	6	1, 1, 2, 6
Insects-i-i	452,044	33	6	1, 1, 2, 6
Insects-i-r-b	79,986	33	6	1, 1, 2, 6
Insects-i-r-i	452,044	33	6	1, 1, 2, 6
Insects-o-o-c	905,145	33	24	4, 10, 14, 24

kNC and Dribble follow different data representations compared to traditional kNN-based methods. While in the kNN instance subsets are buffered with their class labels, kNC and Dribble use summarization strategies and discard the instances, thus resulting in smaller memory consumption and fewer distance computations.

kNC summarizes data using centroids, consequently resulting in a smaller number of distance computations when handling larger data volumes. A centroid is defined as a mean of the instances that are clustered together according to some criteria [26].

The incremental centroids are built by storing the incremental mean of instance attributes. The incremental mean μ_t with the arrival of t values is computed as depicted in Equation 1, where $\bar{x}_{t-1}$ represents the current average, t the number of instances observed thus far, and x_t the arriving value to be incorporated.

$$\mu_t = \frac{\bar{x}_{t-1}(t-1) + x_t}{t} \tag{1}$$

Comparably, the Dribble method also summarizes instances but using Cluster Feature vectors (*CFs*) [30]. *CFs* enable summarizing data in hyperspherical regions and have been used in both data stream clustering and classification techniques [1]. A *CF* is a triplet in the $CF = (N, LS, SS)$ format, where N is the number of instances of the cluster summary, and LS and SS are the linear and square sum of the instances, respectively. Thus, LS and SS are N-dimensional vectors, such that the dimensions match the original features available in instances [30]. From these components, the summary centroid (mean, μ) and its radius (r) are computed according to Equations 2 and 3, respectively, where d is the number of features available.

$$\mu(\mathrm{CF}_i) = \frac{LS_i}{N_i} \tag{2}$$

$$r(\mathrm{CF}_i) = \frac{1}{d}\sum_{i=1}^{d}\sqrt{\frac{N_i(SS_i) - 2(LS_i^2) + N_i(LS_i)}{N_i^2}} \tag{3}$$

In addition to their potential to summarize data, *CFs* also have an additive property, i.e., two feature vectors CF_i and CF_j can be merged by summing their components according to Equation 4 [30]:

$$CF_k = CF_i + CF_j = (N_i + N_j, LS_i + LS_j, SS_i + SS_j) \tag{4}$$

In both kNC and Dribble methods, instances are incorporated into the model by composing a set of n incremental centroids (kNC) or *CFs* (Dribble) in the respective class node, where n is a user-defined hyper-parameter.

To retrieve summary descriptions and build sub-datasets for the prediction step, we perform a top-down traverse in the hierarchy obtaining all centroids/*CFs* stored at the leaf nodes of the tree using the siblings' policy to consider positive instances from the target node and its descendants [24].

The retrieval is straightforward in kNC since the model stores centroids representing a mean instance of the data stream. Meanwhile, in Dribble, we need an additional step to calculate the mean of the *CFs* (cf. Equation 2). Additionally, to avoid noise incorporation due to the intrinsic characteristic of small *CFs*, Dribble implements an outlier control by retrieving only reasonably populated *CFs* (user-defined, by default, one-third of *CF* size).

At any moment, the model can perform a prediction by comparing an unseen instance to the centroids/*CFs* stored at the nodes of the tree. The class prediction is performed by calculating the Euclidean distance between a new instance and the centroids/mean of *CFs* represented in the hierarchy nodes, returning the most frequent label between the selected k-nearest neighbors.

Both methods perform single path and mandatory leaf-node prediction, using a local classifier per node (LCPN) approach [24]. Thus, in the prediction step, we apply one multi-class classifier per class to predict between its child nodes. The resulting predicted label is appended to the final label path, representing the full hierarchical taxonomy predicted to a given instance.

After the prediction step, we update the summary descriptions. kNC and Dribble differ in the way the data is summarized. However, both methods work with a limited size m on centroids/*CFs*, where m is a user-defined hyper-parameter.

On kNC, we incorporate the new (training) instance into the stored centroid by incrementing its average (cf. Equation 1). If the centroid is already full (i.e., if the number of summarized instances equals m), we instantiate a new centroid to the corresponding class node. As a consequence, if a node reaches the stipulated maximum number n of centroids, we perform a forgetting strategy by applying a sliding window and discarding the oldest centroid.

On Dribble, when a new (training) instance is received, we check if the instance is encompassed by any of the hyperspheres represented in the correct class of the instance (i.e., whether the instance is between the *CF* mean and its radius or not). If so, we add the new instance to the respective *CF*. When the number of summarized instances surpasses m, we perform a forgetting strategy by subtracting from the *CF* a statistical description (one mean instance) representing the oldest instance. Otherwise, if the instance is not encompassed by any of the hyperspheres, it starts a new hypersphere (at that moment, yet a single point). If the maximum number (n) of hyperspheres (or points) in a class is reached, the two closest hyperspheres are merged using the additive property of the *CFs*.

Algorithm 1 shows the pseudocode for the implementation of the proposed framework for the automatic identification of disease vector mosquitoes in the context of the hierarchical classification of data streams task.

The algorithm receives a hierarchical data stream *hDS* providing instances $(\vec{x}, \vec{y})$ and the aforementioned hyper-parameters n, m and k, and outputs a predicted label path for each incoming instance representing an entomological taxonomy.

The representation of the taxonomic hierarchy is performed on the first line, and the loop started in line 2 applies the described LCPN approach to retrieve the summary descriptions stored at the tree nodes. The implementation fits both kNC or Dribble methods, except by the data structure retrieved specified in the code as the data from a node. On kNC, data refers to the stored centroids, while on Dribble it refers to the means of the *CFs*.

The prediction step is performed equally for both methods and it is depicted from lines 11 to 15 on the algorithm.

Both methods differ on summary updating and forgetting strategies, represented in the algorithm by line 18. The processes performed in this step on each method separately are described below.

Algorithm 1: Proposed framework for the automatic identification of disease vector mosquitoes.

```
input  : $hDS$ – a hierarchical data stream providing instances $(\vec{x}, \vec{y})$
         $n$ – maximum number of centroids
         $m$ – maximum number of instances to be summarized on a centroid
         $k$ – number of nearest centroids
output : $\hat{\vec{y}}_i$ – a predicted label path for the input instance
1  Tree ← classTaxonomy($hDS$);
2  foreach ($\vec{x} \in hDS$) do
3      predictedNode ← Tree.root;
4      while ¬(predictedNode.isLeaf) do
5          foreach (childNode ∈ predictedNode.children) do
6              targets ← targets ∪ {(childNode.label, childNode.data)};
7              foreach (descendantNode ∈ childNode.descendants) do
8                  targets ← targets ∪ {(childNode.label, childNode.data)};
9              end
10         end
11         foreach (target ∈ targets) do
12             target ← target ∪ {euclideanDistance($\vec{x}$,target.data)};
13         end
14         targets = $(\text{targets})_{1..k}$;
15         predictedNode ← argmax(targets);
16         $\hat{\vec{y}}_i$ ← $\hat{\vec{y}}_i$ ∪ {predictedNode.label};
17     end
18     UpdateSummaryDescription($\vec{y}_i$);
19 end
```

On kNC, the method incorporates the new instance into the stored centroid by incrementing its average or it creates a new centroid if the newest centroid is already full. After that, the method tests whether the number of centroids (n) in the ground-truth node exceeds the stipulated maximum number allowed. If so, it applies a sliding window strategy by forgetting the oldest centroid.

On Dribble, the method finds the nearest *CF* to: update the *CF* using its additive property if the new *CF* is encompassed by the nearest *CF*, or create a new *CF* if it is not encompassed by the nearest *CF*. Then, the method checks if the number of instances represented in the *CFs* stored at the ground-truth node exceeds the stipulated maximum number m allowed. In such a case, a *CF* mean is subtracted from the *CF* to forget a representation of the oldest instance. Finally, the method checks if the number of *CFs* stored at the ground-truth node exceeds the stipulated maximum number allowed. If so, it merges the two closest *CFs* to return to the maximum number n of *CFs* allowed on that node. To this end, a Euclidean distance calculation is performed between all *CFs* at the node.

5 ANALYSIS

This section reports the experimental analysis conducted to compare our proposed framework against existing works in hierarchical data stream classification. First, we provide the experimental protocol adopted. Next, we discuss the results obtained by the framework in terms of prediction and performance.

5.1 Experimental Protocol

In our experiments, we used the 11 hierarchically labeled datasets previously described in Section 4, Table 1.

We compared our proposed framework, instantiated with kNC and Dribble as core methods, to the hierarchical kNN described in Section 3 proposed in [22], hereafter referred to as kNN.

We set up all methods with $n \in \{1, 5, 10, 15, 20\}$, $m \in \{5, 10, 30\}$ and $k \in \{1, 3, 5\}$. The n parameter reflects the buffer size in the kNN method, the number of centroids in the kNC method, and the number of *CFs* in the Dribble method. In addition, m parameter represents the upper bound for the number of instances summarized in a centroid in the kNC and a *CF* in Dribble. Note that kNN does not use the m parameter.

During the experiments, classifiers were assessed in terms of hierarchical F-measure (*hF*) [15]. Like traditional classification metrics, the hierarchical F-Measure is the harmonic mean of its hierarchical precision (*hP*) and hierarchical recall (*hR*) components. The *hP* metric computes the number of labels in a predicted label path that are also components of the ground-truth label path for a given instance. On the other hand, *hR* quantifies the number of ground-truth labels comprehended by the predicted label path for a given instance. Note that label paths represent the mosquitoes' entomological taxonomy from the highest (class Insecta) to the lowest level (mosquitoes' species) of the taxonomic hierarchy.

We report the *hF* metric using the prequential (test-then-train) assessing method, where each instance is used to test the model before it is used for training and updating [4, 11].

Furthermore, we measured the time performance of a model by calculating the number of instances (mosquitoes) that the model can classify per second.

An implementation of the proposed framework comprising both kNC and Dribble methods, as well as an implementation of the tested kNN, are publicly available for reproducibility[2].

Finally, the results obtained by the different models tested were assessed using Friedman [9] and Nemenyi [18] hypothesis tests with a 95% confidence level following the protocol provided in [8].

5.2 Results

In this section, we analyze the results obtained during experimentation. This analysis is divided into three parts: (i) predictive performance assessment, (ii) processing speed comparison, and (iii) statistical validations about the overall behavior of the methods.

5.2.1 Hierarchical F-Measure. Table 2 depicts the hierarchical F-measure (*hF*) obtained by the models tested, where the highest values per dataset are highlighted in bold. These results represent the best *hF* rates and the mean *hF* rates by dataset obtained across the different parameters experimented for each configuration.

Overall, we observe that kNC and Dribble outperform kNN across all datasets. kNC showed an average best *hF* rate of 82.44% and Dribble of 81.80%. The difference from kNC to kNN was 4.87%, and from Dribble to kNN of 3.73%. Concerning the mean *hF* rates, kNC achieved an average rate of 81.26% and Dribble of 79.71%. The difference from kNC to kNN was 5.80%, and from Dribble to kNN of 4.25%.

Besides, kNC and Dribble showed similar *hF* rates in all experiments with small differences favoring kNC (0.64% on the average *hF* and 1.55% on the mean *hF*).

Concerning the best *hF* obtained, kNC achieved higher rates in 10 out of the 11 datasets. Regarding mean *hF* rates, kNC obtained higher rates in all datasets.

[2]http://www.ppgia.pucpr.br/~jean.barddal/datasets/local_knc_dribble.zip

Table 2: *hF* (%) obtained during experiments.

	Best *hF* (%)			**Mean *hF* (%)**		
Datasets	**kNN**	**kNC**	**Dribble**	**kNN**	**kNC**	**Dribble**
Insects-a-b	80.95	**84.37**	83.95	78.97	**83.62**	82.07
Insects-a-i	79.14	**83.02**	82.55	76.32	**81.71**	80.98
Insects-i-a-r-b	79.73	**84.30**	83.76	78.03	**83.50**	82.43
Insects-i-a-r-i	78.52	**82.83**	82.16	75.88	**81.60**	80.83
Insects-i-b	80.05	**84.08**	83.91	78.00	**83.17**	82.69
Insects-i-g-b	84.09	**88.02**	86.75	81.41	**87.12**	85.60
Insects-i-g-i	78.94	83.09	**83.11**	76.42	**81.86**	81.81
Insects-i-i	78.63	**83.29**	83.08	75.91	**81.70**	81.64
Insects-i-r-b	80.87	**84.50**	84.09	78.78	**83.81**	82.04
Insects-i-r-i	78.60	**83.00**	82.71	75.95	**81.64**	81.33
Insects-o-o-c	59.25	**66.32**	63.72	54.34	**64.18**	55.37
Avg. *hF*	78.07	**82.44**	81.80	75.46	**81.26**	79.71
Avg. Ranking	3.00	**1.09**	1.91	3.00	**1.00**	2.00

Furthermore, we appended to Table 2 the average ranking for the best and mean *hF* performances of the methods. One can observe that kNC presents the best results with average rankings of 1.09 and 1.00 in both best and mean *hF* rates. Also, Dribble and kNN have achieved, respectively, the second and third places in both analyses.

Moreover, we evaluate the behavior of the methods over variations of n. Figure 3 compares the average *hF* (%) rates obtained by methods on each variation of n parameter.

The kNN method obtained its better *hF* rates with higher values of n in all datasets. Similarly, kNC also performs better with more stored data (higher n). Meanwhile, Dribble obtained its best mean results with smaller values in n (i.e., $n \in \{1, 5\}$).

We highlight the *hF* rates obtained by Dribble with smaller n values ($n \in \{1, 5\}$, for instance). These results depict that Dribble benefits from smaller numbers of *CFs* (n), thus achieving competitive *hF* rates even using less stored data.

The rationale behind the decrease of *hF* rates with the increase of n on Dribble is related to noise incorporation, as several *CFs* are potentially created to represent a few instances, and thus, are not as representative as those that incorporate most of the data. On the other hand, the performance of Dribble with small data representations is noticeable, as it with $n = 1$ manages to obtain similar or better *hF* rates than the other methods with $n = 20$.

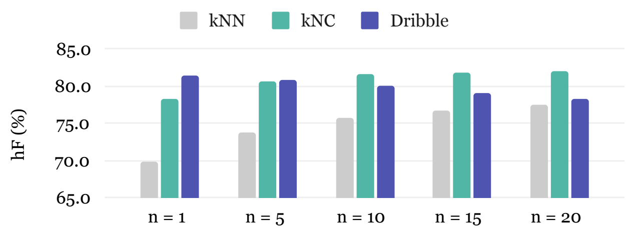

Figure 3: Average *hF* (%) rates obtained by methods when varying the n parameter.

5.2.2 Number of instances processed. Another relevant aspect of the assessment of data streaming approaches regards computational resources. Table 3 reports the number of instances (mosquitoes) classified per second by the methods during the experiments.

The best number of mosquitoes classified per second was obtained by all methods with $n = 1$ since less stored data results also in fewer distance computations needed. The kNN method obtained the higher average number of mosquitoes classified per second considering these best rates (497.09). kNC and Dribble obtained lower rates (496.28 and 487.65) since both methods need to perform extra steps in their learning processes related to the summarization strategies, specifically the creation of centroids in kNC and the creation and merging of *CFs* in Dribble. Still, with $n = 1$, the number of mosquitoes classified per second is similar between all methods.

Averaging all configurations results, the kNC method shows a higher rate than kNN, with a difference of about eight mosquitoes (8.20) favoring the kNC method. Dribble shows an even larger advantage, with a positive difference of about 77.55 mosquitoes to kNC and 85.75 to kNN.

Similar to the analysis conducted in the Hierarchical F-Measure subsection, we appended to Table 3 the average ranking for the number of mosquitoes classified per second of each method. Regarding best rates, kNN obtained first place (1.55) with higher rates in 7 out the 11 datasets, followed by kNC in second place (1.82) and Dribble in third (2.64). However, regarding mean rates, the results are the opposite, since Dribble and kNC achieved the first and second places, respectively, with Dribble obtaining higher mean rates in all datasets.

We also evaluate the processing speed of the methods over variations of n. Figure 4 compares the average number of mosquitoes classified by methods on each variation of n parameter.

We observe that the number of mosquitoes that methods can classify per second decreases according to higher n values. On average, when increasing $n \in \{1, 5, 10, 15, 20\}$, we observe that the number of mosquitoes classified drops by roughly 23% in each step (around 25% for kNN and kNC and around 17% for Dribble). This behavior is expected since larger n values induce larger numbers of distance computations between query instances and stored data.

Table 3: Number of mosquitoes classified per second.

	Best number/sec			Mean number/sec		
Datasets	**kNN**	**kNC**	**Dribble**	**kNN**	**kNC**	**Dribble**
Insects-a-b	502	**523**	515	286	297	**377**
Insects-a-i	**526**	521	512	289	297	**380**
Insects-i-a-r-b	**524**	517	512	285	297	**373**
Insects-i-a-r-i	**526**	518	510	290	296	**380**
Insects-i-b	511	**537**	503	290	288	**365**
Insects-i-g-b	**511**	496	510	286	304	**376**
Insects-i-g-i	**528**	518	513	287	300	**378**
Insects-i-i	525	**528**	505	292	294	**378**
Insects-i-r-b	**525**	521	510	289	298	**373**
Insects-i-r-i	**527**	517	507	290	296	**379**
Insects-o-o-c	263	263	**268**	144	150	**211**
Avg. number	**497.09**	496.28	487.65	275.27	283.47	**361.02**
Avg. Ranking	**1.55**	1.82	2.64	2.91	2.09	**1.00**

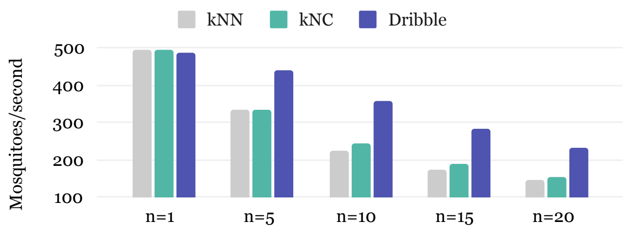

Figure 4: Average number of mosquitoes classified per second by methods when varying the n parameter.

Also, it is expected that this increase affects Dribble less since it uses an outlier control (cf. Section 4) and thus do not perform the same amount of distance computations as kNN or kNC.

5.2.3 Statistical validation. We have conducted a statistical analysis to determine whether the difference between kNN, kNC, and Dribble is significant or not. To that, we applied Friedman and Nemenyi statistical tests. We used as statistical sample sets all the hF and mosquitoes classified per second rates obtained by methods in two separate analyses.

First, the Friedman test showed a significant difference between the methods in both hF ($p\text{-}value = 2.82 \times 10^{-44}$) and mosquitoes classified per second rates ($p\text{-}value = 1.55 \times 10^{-11}$). After that, we applied the Nemenyi test to perform pairwise comparisons.

Figure 5 shows the resulting two critical difference charts for the hF rates (a) and the number of mosquitoes classified per second (b) obtained by kNN, kNC, and Dribble methods. The Nemenyi tests showed statistical differences between all methods in both analyses.

These results depict that kNC can obtain significantly better hF rates than the other methods, and Dribble can also obtain significantly better hF rates than kNN. Furthermore, when considering scenarios where data changes are less severe, and the entire data stream contains relevant information, the kNN method requires a memory buffer big enough to consider all concepts together in the

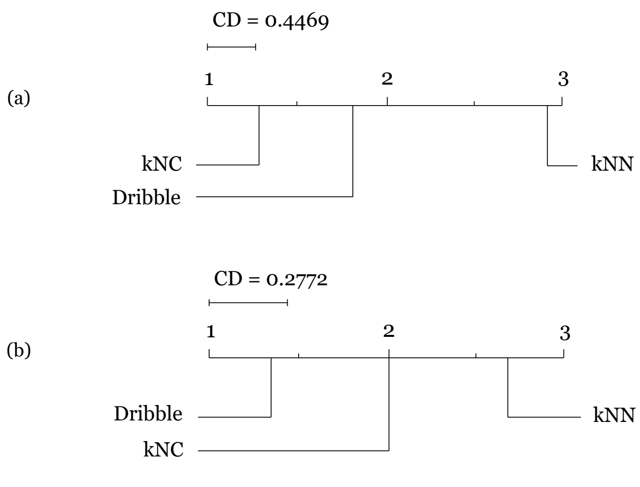

Figure 5: Critical differences chart for (a) hF rates and (b) mosquitoes classified per second rates.

sampled instances. However, this strategy may become infeasible due to computational resource constraints of specific scenarios. The same cannot be said for kNC and Dribble, which are capable of summarizing the entire data stream via centroids and *CFs*, thus enabling a representation of the data using all training instances and not putting computational performance in jeopardy.

Regarding speed comparison, Dribble is significantly faster than the other methods, and kNC is yet significantly faster than kNN. Here, it is important to highlight that both kNC and Dribble methods use additional steps in their learning processes in order to apply data summarization strategies. At first glance, this might indicate that both methods could be slower compared to the kNN method. However, the summarization property itself reverses this difference by summarizing data inside a centroid or a *CF*. Note that even using equal values in n, kNC and Dribble methods manage to perform fewer distance computations within their data representations through the summarization obtained by the parameter m while the centroid or the *CF* is not full.

Combining the analysis of prediction quality and computational resources, we observe that kNC and Dribble methods have the advantage of summarizing information with different granularity levels depending on the problem, making them more versatile than the traditional hierarchical kNN method. Also, considering the better results obtained by kNC in terms of hF and by Dribble regarding processing time, the best setup depends on specific data distribution characteristics and available resources.

6 CONCLUSION

In this study, we proposed a framework for the automatic identification of disease vector mosquitoes in the context of the Hierarchical classification of data streams area as a helping tool to the control of Vector-borne diseases.

At the framework core, we introduced k-Nearest Centroids and Dribble, two novel methods for the hierarchical classification of data streams fitted to hierarchical data streams representing disease vector mosquitoes and their entomological taxonomy. Both schemes rely on summarization techniques to represent data and work with constrained memory usage and time, being a valuable option to act as a base method for vector disease mosquitoes' classification in traps based on machine learning.

We also provide an adapted hierarchical version of a disease vector mosquitoes dataset, including hierarchical label paths describing the taxonomy of mosquitoes.

Our framework could classify more mosquitoes per second and also with better prediction rates when compared to existing state-of-the-art kNN-based techniques.

A resulting implementation of the proposed framework and the adapted hierarchical dataset are publicly available for download to be used as a baseline to further research on the topic, such as computational resources analysis, concept drift detection, and other data summarizing approaches.

REFERENCES

[1] Charu C Aggarwal, Jiawei Han, Jianyong Wang, and Philip S Yu. 2006. A framework for on-demand classification of evolving data streams. *IEEE Transactions on Knowledge and Data Engineering* 18, 5 (2006), 577–589.

[2] David W Aha, Dennis Kibler, and Marc K Albert. 1991. Instance-based learning algorithms. *Machine learning* 6, 1 (1991), 37–66.

[3] Jean Paul Barddal, Heitor Murilo Gomes, Fabrício Enembreck, Bernhard Pfahringer, and Albert Bifet. 2016. On dynamic feature weighting for feature drifting data streams. In *Joint european conference on machine learning and knowledge discovery in databases*. Springer, 129–144.

[4] Albert Bifet and Richard Kirkby. 2009. Data Stream Mining: A Practical Approach.

[5] Maha Bouzid, Julii Brainard, Lee Hooper, and Paul R Hunter. 2016. Public health interventions for Aedes control in the time of Zikavirus–A meta-review on effectiveness of vector control strategies. *PLoS neglected tropical diseases* 10, 12 (2016), e0005176.

[6] Liang Cao, Yufeng Wang, Bo Zhang, Qun Jin, and Athanasios V Vasilakos. 2018. GCHAR: An efficient Group-based Context—Aware human activity recognition on smartphone. *J. Parallel and Distrib. Comput.* 118 (2018), 67–80.

[7] Daniel da Silva Motta, Roberto Badaró, Alex Santos, and Frank Kirchner. 2018. Use of Artificial Intelligence on the Control of Vector-Borne Diseases. *Vectors and Vector-Borne Zoonotic Diseases* (2018).

[8] Janez Demšar. 2006. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research* 7, Jan (2006), 1–30.

[9] Milton Friedman. 1937. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *J. Amer. Statist. Assoc.* 32, 200 (1937), 675–701.

[10] Joao Gama. 2010. *Knowledge discovery from data streams*. Chapman and Hall/CRC.

[11] João Gama, Raquel Sebastião, and Pedro Pereira Rodrigues. 2013. On evaluating stream learning algorithms. *Machine learning* 90, 3 (2013), 317–346.

[12] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. 2014. A survey on concept drift adaptation. *ACM computing surveys (CSUR)* 46, 4 (2014), 44.

[13] Kun-Yi Huang, Chung-Hsien Wu, Qian-Bei Hong, Ming-Hsiang Su, and Yi-Hsuan Chen. 2019. Speech Emotion Recognition Using Deep Neural Network Considering Verbal and Nonverbal Speech Sounds. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 5866–5870.

[14] Ananya Joshi and Clayton Miller. 2021. Review of machine learning techniques for mosquito control in urban environments. *Ecological Informatics* (2021), 101241.

[15] Svetlana Kiritchenko and Fazel Famili. 2005. Functional Annotation of Genes Using Hierarchical Text Categorization. *Proceedings of BioLink SIG, ISMB* (2005).

[16] Aris Kosmopoulos, Ioannis Partalas, Eric Gaussier, Georgios Paliouras, and Ion Androutsopoulos. 2015. Evaluation measures for hierarchical classification: a unified view and novel approaches. *Data Mining and Knowledge Discovery* 29, 3 (2015), 820–865.

[17] Moritz UG Kraemer, Marianne E Sinka, Kirsten A Duda, Adrian QN Mylne, Freya M Shearer, Christopher M Barker, Chester G Moore, Roberta G Carvalho, Giovanini E Coelho, Wim Van Bortel, et al. 2015. The global distribution of the arbovirus vectors Aedes aegypti and Ae. albopictus. *elife* 4 (2015), e08347.

[18] Peter Nemenyi. 1962. Distribution-free multiple comparisons. In *Biometrics*, Vol. 18. International Biometric Society, 263.

[19] Hai-Long Nguyen, Yew-Kwong Woon, and Wee-Keong Ng. 2015. A survey on data stream clustering and classification. *Knowledge and information systems* 45, 3 (2015), 535–569.

[20] Liadan O'callaghan, Nina Mishra, Adam Meyerson, Sudipto Guha, and Rajeev Motwani. 2002. Streaming-data algorithms for high-quality clustering. In *Proceedings 18th International Conference on Data Engineering*. IEEE, 685–694.

[21] World Health Organization et al. 2014. *A global brief on vector-borne diseases.* Technical Report. World Health Organization.

[22] Antonio Rafael Sabino Parmezan, Vinicius MA Souza, and Gustavo EAPA Batista. 2018. Towards Hierarchical Classification of Data Streams. In *Iberoamerican Congress on Pattern Recognition*. Springer, 314–322.

[23] Sergio Ramírez-Gallego, Bartosz Krawczyk, Salvador García, Michał Woźniak, and Francisco Herrera. 2017. A survey on data preprocessing for data stream mining: Current status and future directions. *Neurocomputing* 239 (2017), 39–57.

[24] Carlos N Silla and Alex A Freitas. 2011. A survey of hierarchical classification across different application domains. *Data Mining and Knowledge Discovery* 22, 1-2 (2011), 31–72.

[25] V. M. A. Souza, D. M. Reis, A. G. Maletzke, and G. E. A. P. A. Batista. 2020. Challenges in Benchmarking Stream Learning Algorithms with Real-world Data. *Data Mining and Knowledge Discovery* (2020), 1–54.

[26] Michael Steinbach, Levent Ertöz, and Vipin Kumar. 2004. The challenges of clustering high dimensional data. In *New directions in statistical physics*. Springer, 273–309.

[27] Eduardo Tieppo, Roger Robson dos Santos, Jean Paul Barddal, and Júlio Cesar Nievola. 2021. Hierarchical classification of data streams: a systematic literature review. *Artificial Intelligence Review* (2021), 1–40.

[28] Alexey Tsymbal. 2004. The problem of concept drift: definitions and related work. *Computer Science Department, Trinity College Dublin* 106, 2 (2004), 58.

[29] Feihong Wu, Jun Zhang, and Vasant Honavar. 2005. Learning classifiers using hierarchically structured class taxonomies. In *International Symposium on Abstraction, Reformulation, and Approximation*. Springer, 313–320.

[30] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. 1996. BIRCH: an efficient data clustering method for very large databases. *ACM sigmod record* 25, 2 (1996), 103–114.