Hierarchical and Non-Hierarchical Classification of Transposable Elements with a Genetic Algorithm

Gean T. Pereira, Ricardo Cerri

Federal University of São Carlos, São Carlos, Brazil
{gean.pereira, cerri}@ufscar.br

Abstract. In traditional classification an instance is assigned to one class within a small set of classes. However, there are problems where an instance is related to many classes hierarchically structured, known as Hierarchical Classification (HC), which is present in many domains like Text Categorization, Music Genre Classification and Bioinformatics. A topic that has gained attention recently is the classification of Transposable Elements (TEs), which are DNA sequences capable of moving inside the genome. TEs have a great importance in the genetic variability of species, since they can modify the functionality of host genes. Despite the research relevance, just a few tools perform its automatic classification and most of them do not use more elaborated strategies, like using Machine Learning to learn models from data. Moreover, the interpretability of these methods is still an issue. In this work, we extend the original study that proposed the global method HC-GA, presenting some improvements such as new fitness functions which were compared and analyzed per level and in the leaf nodes, as well as with respect to the interpretability of the rules generated. Besides, we compare the new best results of HC-GA with the results of flat methods previously reported in the original paper for the task of predicting the TEs leaf node classes. As experiments showed, flat methods do not performed well for HC datasets, even ignoring the hierarchical relationships among classes. We believe this occurred due the high imbalance of these datasets, which is something that flat methods do not handle well, unlike HC ones. HC-GA overcame flat classifiers, presenting promising results for multiple class levels including leaf node classes, even though it was not originally designed for this purpose, and considering the difficulty of predicting lower classes in a hierarchy.

Categories and Subject Descriptors: H.2.8 [**Database Applications**]: Data Mining; I.2.6 [**Artificial Intelligence**]: Learning

Keywords: Genetic Algorithms, Hierarchical Classification, Machine Learning, Rule Induction, Transposable Elements

1. INTRODUCTION

In Machine Learning (ML), most of the related works on classification tasks addresses Flat Classification, where an instance from a dataset is associated with one class within a usually small set of classes [Mitchell et al. 1997]. However, there are more complex problems where instances can be assigned to many classes simultaneously, within dozens or even hundreds of classes arranged in a hierarchical structure, which has certain constraints. This kind of classification is known in the literature of ML as Hierarchical Classification (HC) [Silla and Freitas 2011].

One of the major areas for applying HC methods is Bioinformatics, which as already pointed out in the literature, when used in combination with ML often results in better predictions compared to other approaches [Loureiro et al. 2013]. In this context, a topic that has gained attention in the last years is the study of Transposable Elements (TEs), which are fragments of DNA capable of moving within the genome of their hosts [Biémont 2010; Jurka et al. 2005; McClintock 1993]. According to recent research, TEs are responsible for mutations in a variety of organisms, including the human genome, being responsible for the genetic variability of many species [Kazazian 2004; Rebollo et al. 2012; van de Lagemaat et al. 2003]. Thus, the correct classification of these elements brings benefits to the understanding of several species as well as how their evolution has occurred, which are important aspects for the field of Biology and Bioinformatics [Feschotte 2008; Wheeler et al. 2012].

Despite the relevance of creating classification models for TEs, there are just a few tools in the literature which can actually perform their automatic classification [Altschul et al. 1990; Smit et al. 1996; Steinbiss et al. 2009; Abrusán et al. 2009; Feschotte et al. 2009; Hoede et al. 2014]. However, most of these tools do not use more elaborated strategies, like using ML to generate models from data. Also, none of them has treated the TEs classification task with HC, which can bring more semantics to the classification problem, like in terms of classes relationships information, which is very useful for biologists. Moreover, the lack of interpretability in these methods is still an open issue.

This paper presents an extension of the work proposed in [Pereira and Cerri 2017], which presented a new global hierarchical method based on a Genetic Algorithm (GA) called HC-GA, that is capable of generating and evolving HC rules. In this current work, we improve the HC-GA by implementing a variety of new fitness functions that allow new best results for the TEs hierarchical datasets. In addition, more elaborate experiments were performed as an analysis per level and an interpretability analysis of the models generated by each fitness used. Besides, the results from the previous study were included, like the ones with well established ML methods, the aim of which is to verify the potential of flat strategies predicting the leaf node classes of TEs. Even though the hierarchical method HC-GA was not specifically designed to predict leaf classes, we compare its performance with flat methods in order to check if there are problems with these operations. Also, we verify the predictions made by HC-GA in several levels of the TEs class hierarchy since in HC problems, obtaining good accuracy in multiples levels is a challenging and important task.

To summarize, the contributions of this work are as follows:

—Implementation of a variety of new fitness functions in HC-GA, some of them more focused on accuracy and others on the model's interpretability;
—More experiments with HC-GA were performed using the new fitness functions, including an analysis per level and for the particular case of leaf class prediction, besides the interpretability analysis of the sets of rules generated;
—A comparison of the new best results for the proposed hierarchical method with some non-hierarchical methods in the challenging task of predicting the leaf classes of a hierarchy.

2. BACKGROUND

Classification is a popular ML task whose goal is to assign instances to predefined categories. The algorithms are given as input a set of N d-dimensional training instances $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N\}$, as well as their set of class labels $C = \{C_{\mathbf{x}_1}, C_{\mathbf{x}_2}, ..., C_{\mathbf{x}_N}\}$, in which $C_{\mathbf{x}_i} \in \{C_1, ...C_k\}$ in a k-class problem. The majority of classification problems associates each instance to a single class within an often small set of classes, meaning that $C_{\mathbf{x}_i}$ is a single categorical value in $\{C_1, ...C_k\}$. This particular problem is regarded as *Flat* or *non-Hierarchical* Classification.

However, there are more complex classification problems in which the classes are organized according to a hierarchical structure, and each instance may be simultaneously associated to multiple classes in this hierarchy. As an example, consider the hierarchy shown in Figure 1. Each node represents a class, thus node "1/1/2" is a subclass of "1/1", which in turn is a subclass of "1", forming a complete path through the hierarchy. This particularly task consists of assigning a hierarchical path to a given instance and is known as Hierarchical Classification (HC).

The set of class labels for a HC problem can be represented as a matrix $\mathbf{V} = \{\mathbf{v}_{\mathbf{x}_1}, \mathbf{v}_{\mathbf{x}_2}, ..., \mathbf{v}_{\mathbf{x}_N}\}$, in which $\mathbf{v}_{\mathbf{x}_i}$ is the c-dimensional binary class vector associated with instance $\mathbf{x}_i$, in a hierarchy with c nodes (classes). Each position of the class vector $\mathbf{v}_{\mathbf{x}_i}$ represents a class, that is set to 1 in case of $\mathbf{x}_i$ is assigned to that respective class, or 0 otherwise.

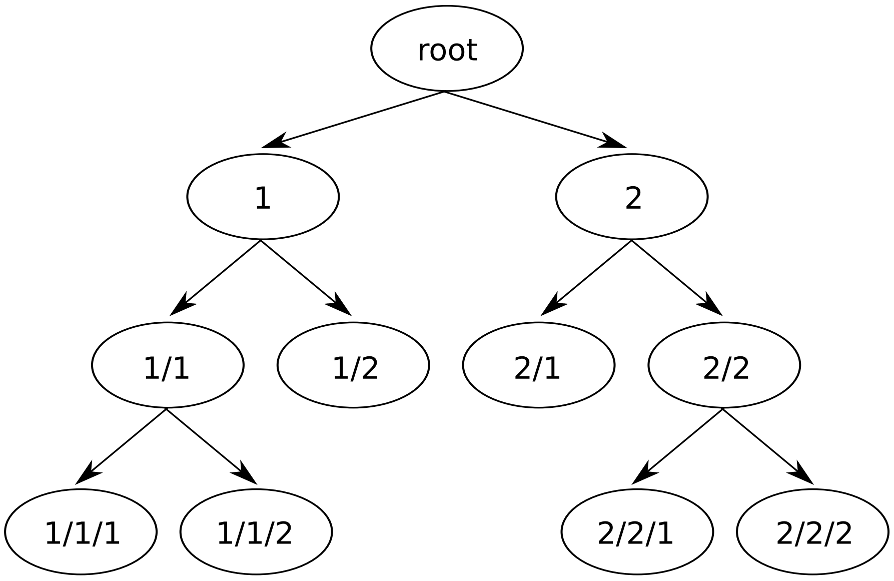


Fig. 1. Example of a tree class hierarchy.

Several other peculiarities compose this kind of classification, such as the type of the hierarchy structures used, the different depths in which a classification can be performed, the exploration strategies applied to a class hierarchy and the evaluation metrics employed. Regarding the structures, two commonly used ones are Trees and Directed Acyclic Graphs (DAG), differing from each other in the number of paths that a classifier can explore in the structure. In respect of how deep a classification is performed, there are two main categories: (i) Mandatory Leaf-Node Prediction, when the classifier's output is always a leaf node, and (ii) Non-Mandatory Leaf-Node Prediction, which the most specific class predicted for a given example could be a node at any level of the hierarchy.

There also different exploration approaches to deal with HC problem, called *local* and *global* [Silla and Freitas 2011]. In the training phase of a global approach, a single classifier is induced to deal with an entire hierarchy of classes, while in the test phase, the classification of a new instance occurs in a single step [Vens et al. 2008]. However, the local approach have a training modularity, which means that local information about classes is continuously used to build the models, whilst the test phase often follows a top-down strategy to classify new instances, in which the hierarchy is explored towards the leaves, level by level [Costa et al. 2007]. Another difference between these strategies is the type of algorithms used. In the local approach, flat (and well established) classification algorithms such as Decision Trees, Bayes classifiers and Support Vector Machines can be trained, without any adaptation, to produce a hierarchy of predictions. Although in the global approach, since it induces a single classifier to handle all the particularities of a HC problem, it does not usually use conventional algorithms, unless they are heavily adapted to consider a hierarchy of classes.

3. RELATED WORK

In [Ghazi et al. 2010], the HC task is explored in the context of automatic text classification through emotions expressed in text. The authors propose a method that organizes hierarchically the emotions presented in text according to their neutrality or polarity. This method is validated in two datasets along with a flat method, which proved to be worse than the hierarchical one. Another interesting analysis described in this work, was that the presence of a high imbalance in the datasets, which usually has an important impact on the performance of classifiers, is smoothed in hierarchical approaches.

In [Zimek et al. 2010], comparisons are made with multi-class classifiers that explore hierarchical information and flat classifiers, such as SVMs, Decision Trees and Ensembles. The datasets adopted are related to protein enzymes classification and synthetic data. Despite the hierarchical nature of the problem addressed, experiments performed showed that the exploration of hierarchical information does not always give better results. According to the authors, exploring class hierarchies improves the prediction performance in case of synthetic data, which is not true for protein enzyme classification.

In [Silla and Kaestner 2013], authors address the HC of bird species using data about their audio recordings. They apply three approaches, two hierarchical (local and global) and one flat classification approach. For the flat and local approaches, a classic Naive Bayes was employed, while for the global approach they used a global version of the Naıve Bayes algorithm. They employed a standard scientific taxonomy of birds as the hierarchical structure, and the features from the bird songs were obtained by computing several acoustic quantities from intervals of the audio signal. The experiments were evaluated using the hierarchical F-measure metric, and results showed that the use of a global hierarchical approach outperforms both flat and local approaches. For last, they recommend the use of global models, which in their words, could be a feasible way to improve the classification performance for problems with a large number of classes.

In [Pereira and Cerri 2017], a global hierarchical method is proposed, based on a Genetic Algorithm to induce HC rules that classify Transposable Elements in multiple levels of their hierarchy. That method is called HC-GA (**H**ierarchical **C**lassification with a **G**enetic **A**lgorithm), and along with some popular flat classifiers, was applied to classify two new hierarchical TEs datasets suitable for ML methods. The main purpose of that paper was to compare the power of non-hierarchical classifiers to correctly predict TEs leaf node classes, comparing them with the proposed method.

4. GLOBAL RULE INDUCTION METHOD

In this section, the global method for HC proposed in [Pereira and Cerri 2017] which is called HC-GA (**H**ierarchical **C**lassification with a **G**enetic **A**lgorithm), is presented with more details about its genetic operators and the new fitness functions implemented. This method generates a set of interpretable HC rules that classifies instances at multiples levels of a hierarchy. The evolutionary process of inducing rules is detailed in Algorithm 1.

Along with the evolutionary process (lines 8 to 20), Algorithm 1 presents, on the top of it, a sequential covering strategy to induce rules that classify instances from a hierarchical dataset (lines 2 to 7 AND lines 21 to 26). During this procedure, HC rules are induced (line 21) and used to cover instances, which are immediately removed from the training set (line 22). This allows the generation of new rules that will cover the remaining instances when the evolutionary process is restarted.

4.1 Solution Representation and Encoding

When GAs are adopted for rule induction, two frequently used approaches to represent rules are the Pittsburgh [Smith 1980] and Michigan approaches [Holland 1986]. In the Pittsburgh approach, each individual corresponds to a set of rules, whereas in the Michigan approach, each individual represents a single rule, and a set of rules is represented by an entire population of individuals [Srinivasan and Ramakrishnan 2011; Otero et al. 2013]. In this work, the Michigan approach is used.

An individual in HC-GA is a fixed-length coded vector containing real values that represents the antecedent of a classification rule. Each test within the antecedent is related to an attribute from a dataset, and its possible components are a FLAG, an operator (OP) and an attribute index or value (values Δ). Thus, each set of four positions of this vector represents one test, being a 4-tuple{FLAG, OP, Δ_1, Δ_2}. Figure 2 illustrates an individual and its 4-tuples.

Algorithm 1 HC-GA main loop.

```
Require: Training dataset $D$, number of generations $G$, population size $p$, minimum number of covered instances per rule $minCov$, maximum number of covered instances per rule $maxCov$, maximum number of uncovered instances $maxUncov$, crossover rate $cr$, mutation rate $mr$, tournament size $t$, number of individuals selected by elitism $e$, probability of using a test in a rule $pt$.
1: $Rules \leftarrow \emptyset$
2: while $|D| > maxUncov$ do
3:   $currentPop \leftarrow generatePopulation(D, p, pt)$
4:   $currentPop \leftarrow localSearch(minCov, maxCov, currentPop)$
5:   $calculateFitness(currentPop, D)$
6:   $bestRule \leftarrow getBestRule(currentPop)$
7:   $j \leftarrow 0$
8:   while $j < G$ or $ruleConvergence()$ do
9:     $newPop \leftarrow \emptyset$
10:    $newPop \leftarrow newPop \cup elitism(currentPop, e)$
11:    $parental \leftarrow tournamentSelection(currentPop, t)$
12:    $offspring \leftarrow uniformCrossover(parental, cr)$
13:    $offspring \leftarrow mutation(offspring, mr)$
14:    $newPop \leftarrow newPop \cup offspring$
15:    $newPop \leftarrow localSearch(minCov, maxCov, newPop)$
16:    $currentPop \leftarrow newPop$
17:    $calculateFitness(currentPop, D)$
18:    $bestRule \leftarrow getBestRule(currentPop, bestRule)$
19:    $j \leftarrow j + 1$
20:  end while
21:  $Rules \leftarrow Rules \cup bestRule$
22:  Remove instances from $D$ covered by $bestRule$
23: end while
24: if $|D| > 0$ then
25:   Remove instances from $D$ using the $defaultRule$
26: end if
```

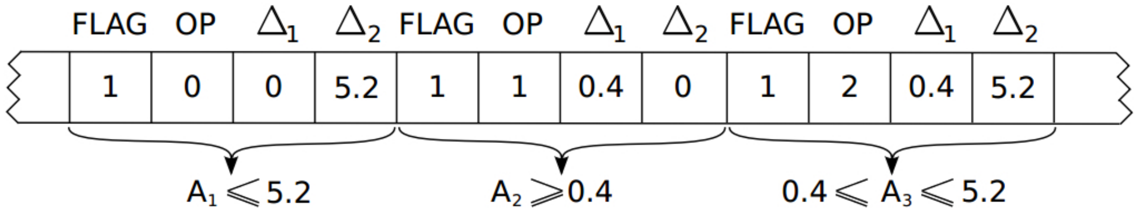


Fig. 2. Example of an encoded individual. Adapted from [Cerri et al. 2014]

The FLAG gene in the encoded vector can receive the value 0 or 1, indicating the absence or presence of the corresponding test in a rule's antecedent, respectively (activated or not). That allows rules to have different numbers of tests. The OP gene defines one of the possible operators to use in the test of a given attribute (numerical/ordinal or categorical), which is set randomly. In turn, the Δ_1 and/or Δ_2 genes receive real numbers values used as test conditions.

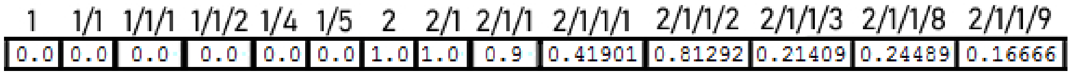
HC-GA is capable of working with categorical and numerical attributes. Since in this work we adopted datasets containing only numerical attributes, our method used only the operators $\leq$ and $\geq$. In case of choosing randomly a value 0 for OP, the operator $\leq$ is used in the test, making Δ_1 be set to 0 and Δ_2 to receive the value of the attribute being tested. Otherwise, choosing the value 1 the operator

$\geq$ is used, and the opposite occurs with the Δ genes. Also, it is possible to check whether a numeric attribute value belongs to a certain range when a value 2 is chosen, such as $\Delta_1 \leq A_k \leq \Delta_2$, where Δ values are the lower and upper bounds for numeric attributes in a test. In that case, the values of Δ are randomly chosen so that the value of the test condition satisfies the instance's attribute. It is important to emphasize that all the possible operators are previously indexed with fixed indexes, enabling a test to be easily assembled through the corresponding indexes and operators.

4.2 Consequent Construction and Rule Format

Once an antecedent is built, is possible to calculate its mean class vector using the Equation 1. This vector is obtained by dividing the set of instances covered by rule r that is classified in class i ($S_r\,i$) by the set of instances covered by r (S_r). Thus, each of its i_{th} positions is a real value between [0, 1] that represents the probability of a given instance belonging to a class i in the hierarchy. In other words, for instances that satisfy the antecedent of r, this vector can be interpreted as the consequent of r. Figure 3 shows an example of a consequent vector, where each position contains the real value that represents the probability of association of a given class to an instance, always maintaining the hierarchical consistency of the classes.

$$\overline{\mathbf{v}}_{r\,i} = \frac{|S_{r\,i}|}{|S_r|} \tag{1}$$



1	1/1	1/1/1	1/1/2	1/4	1/5	2	2/1	2/1/1	2/1/1/1	2/1/1/2	2/1/1/3	2/1/1/8	2/1/1/9
0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.9	0.41901	0.81292	0.21409	0.24489	0.16666

Fig. 3. Example of a rule's consequent.

Moreover, a threshold is applied in the consequent vectors in order to obtain the final predictions of HC-GA. We are using a threshold of 0.5, so if the i_{th} value in the vector is higher than or equal to 0.5, it receives the value 1. Otherwise, the position is set to 0.

4.3 Population of Rules

A population of rules is created by a seeding process, in which training instances are randomly selected and its attributes are used to create individuals. Each attribute from an instance has a *pt* probability of being used in the antecedent that will be generated, meaning that this *pt* probability is used to enable/disable the FLAG genes. Usually, a low value is chosen for both *pt* in the seeding process and mutation of the FLAGs, since a high value would result in rules with many active tests which would cover just a few instances or only the seed instance.

The seeding process ensures that each individual covers at least one training instance (its own seed instance), and an instance is considered covered if all the rule's active tests are satisfied by the instance attributes. This process is repeated until a population with the desired size is generated.

4.4 Fitness Function

A variety of fitness functions has been implemented and tested in HC-GA, such as the following:

1) Hierarchical Evaluation Metrics (hP, hR and hF): The hierarchical versions of the conventional Precision, Recall and F1 Score metrics, that were proposed to take into account the hierarchical relationships among classes [Kiritchenko et al. 2005]. Their formal definition is shown in Equations 2, 3 and 4, where C_i and Z_i are the sets of true and predicted classes for an instance i, respectively.

$$hP = \frac{\sum_i |Z_i \cap C_i|}{\sum_i |Z_i|} \quad (2)$$

$$hR = \frac{\sum_i |Z_i \cap C_i|}{\sum_i |C_i|} \quad (3)$$

$$hF = \frac{2\,.\,hP\,.\,hR}{hP + hR} \quad (4)$$

2) Area Under the Precision Recall Curve ($AU(\overline{PRC})$): For each class, HC-GA outputs real values in the interval [0, 1] that can be interpreted as the probability of assignment. This allows the calculation of Precision-Recall curves (PR-curves) [Davis and Goadrich 2006], which are obtained by applying multiple threshold values in the outputs of HC-GA, resulting in different Precision and Recall values (one for each threshold), in other words, points within the PR space. Thus, the union of these points forms a PR-curve, making possible to calculate the Area Under the Curve (AUC).

To calculate the area under the PR-curve, an interpolation of the Precision-Recall points (PR-points) and posterior connection is required. Connecting the points without interpolation would artificially increase the AUC. Applying a threshold, a PR-point is obtained through Equations 5 and 6, which are the micro-average of Precision and Recall. In these equations, i ranges from 1 to number of classes $|C|$, whereas the number of True Positives, False Positives, and False Negatives are represented by TP, FP, and FN, respectively. With the PR-points, it is possible to calculate the AU($\overline{PRC}$) [Vens et al. 2008], where its value ranges between [0, 1] which the higher, the better.

$$\overline{P} = \frac{\sum_i TP_i}{\sum_i TP_i + \sum_i FP_i} \quad (5) \qquad \overline{R} = \frac{\sum_i TP_i}{\sum_i TP_i + \sum_i FN_i} \quad (6)$$

3) Percentage Coverage (PC): As its name suggests, this fitness measures the percentage of instances covered by a given rule. Thus, this value is obtained by Equation 7, where S_r is the set of instances covered by a rule r and S is the total set of instances.

$$PC(r) = \frac{|S_r|}{|S|} \quad (7)$$

4) Variance Gain (VG): This fitness measures the decrease of variance achieved among instances that are covered by a rule and instances that are not. This means that VG partitions the set of instances in sets of similar instances. The VG formula [Vens et al. 2008] is shown in Equation 8. According to this equation, the training set S is divided into two subsets: (1) the set of instances covered by rule r (S_r) and (2) the set of instances not covered by rule r ($S_{\neg r}$). The VG of a rule r is then calculated with respect to set S. The calculation also involves the Variance (var) of the set of instances covered and not covered by rule r, which its formula is presented in Equation 9.

$$VG(r, S) = var(S) - \frac{|S_r|}{|S|} \cdot var(S_r) - \frac{|S_{\neg r})|}{|S|} \cdot var(S_{\neg r}) \quad (8)$$

$$var(S) = \frac{\sum_{k=1}^{|S|} weightedEuclideanDistance(\mathbf{v}_k, \overline{\mathbf{v}})^2}{|S|} \quad (9)$$

Variance is defined as the mean square distance between two mean class vectors, being that from each of the training instances and from the set of instances being considered (S, S_r or $S_{\neg r}$). To calculate this distance, the weighted Euclidean Distance presented in Equation 10 is used, which adopts a weighted scheme to handle the particularities of a hierarchical structure. In this equation, w_i corresponds to the weight associated to the i^{th} class in the hierarchy, while v_{1i} and v_{2i} are the values associated to the i^{th} position in the mean class vectors $\overline{\mathbf{v}}_1$ and $\overline{\mathbf{v}}_2$, respectively. Weights are associated to all classes at all levels, because, in the context of HC, similarities between classes at higher levels are often more relevant than the ones at deeper levels of a hierarchy.

$$weightedEuclideanDistance(\overline{\mathbf{v}}_\mathbf{1}, \overline{\mathbf{v}}_\mathbf{2}) = \sqrt{\sum_{i=1}^{|C|} w_i \cdot (\overline{v}_{1i} - \overline{v}_{2i})^2} \tag{10}$$

$$w_i = w_0 \times \sum_{j=1}^{P_i} w(p_j)/P_i \tag{11}$$

We use the same weighted scheme adopted in [Vens et al. 2008], where, after experiments, the authors reported that the best scheme is defined according to Equation 11. The weight w_0 associated to a class at the first level is defined as *0.75*, whereas the weight of a class i is recursively defined as the multiplication of w_0 and the mean weight of all its ancestor classes P_i.

These fitness functions were chose for many reasons. Using hP, hR and hF metrics as functions, our aim was to directly maximize the performance of these hierarchical metrics in the final results. Regarding the use of AU($\overline{PRC}$), this is justified by reason of the many possible threshold values that can be applied to the final outputs of HC-GA. During the search for good rules it is often difficult to evaluate rules based on a fixed threshold, since a bad choice for a threshold may lead to many rules with a poor performance. Since finding an "optimal" threshold is a difficult task, in which a low threshold leads to many classes being assigned to instances, and this results in a high recall and low precision, occurring the opposite when a threshold is high, we chose AU($\overline{PRC}$) to overcome this problem and avoid losing good rules during the search.

However, unlike hierarchical metrics and AU($\overline{PRC}$) whose aim is the rule's accuracy, functions VG and PC were adopted in order to generate a small set of rules, maximizing the models interpretability. As shown, VG measures the decrease of Variance achieved among instances that are covered by a rule and instances that are not, this means that VG partitions the data into similar sets. We chose this function in order to guide the search for rules with high cohesion. Regarding the PC function, the purpose of its use was to search for more general rules that cover large portions of a dataset.

4.5 Genetic Operators

In this sub-section we describe the genetic operators applied during the evolution of HC rules. After calculating the population fitness (Algorithm 1, line 5) and starting the evolutionary process (Algorithm 1, lines 8 to 20), an *Elitism* operator is applied to the best e rules from the current population (Algorithm 1, line 10), saving and passing them directly to the new population. Thus, a set of $p - e$ parent rules (where p is the population size) is selected using *Tournament Selection* [Wetzel 1983] (Algorithm 1, line 11), which enables the *Uniform Crossover* (Algorithm 1, line 12) to be applied between these rules, thus generating the offspring rules. This crossover is presented in Algorithm 2, being a specialized crossover created to combining similar rules.

The idea behind this operator is that similar rules probably cover instances close to each other in the search space, meaning that they are probably classified in the same classes too, then combining them could lead to rules with high cohesion and more accuracy being generated. This operator

Algorithm 2 Specialized crossover.

```
Require: Parent rules $R$, crossover rate $cr$.
1: $offsprings \leftarrow \emptyset$
2: while $|R| >= 2$ do
3:     $parent_1 \leftarrow R[0]$
4:     $R \leftarrow R - parent_1$
5:     $parent_2 \leftarrow getMostSimilarRule(parent_1, R)$
6:     $R \leftarrow R - parent_2$
7:     $offsprings \leftarrow offsprings \cup UniformCrossover(parent_1, parent_2, cr)$
8: end while
9: return $offsprings$
```

takes the distances between the mean class vectors of two rules to determine if they are similar or not (Algorithm 2, line 5), which the lower the Euclidean distance, the more similar are the rules. After that, a conventional uniform crossover is applied to those rules according to a crossover rate *cr* (Algorithm 2, line 7). This process is repeated until a whole new generation of rules is obtained.

Once the crossover is finished, a new population of rules is submitted to a mutation operator (Algorithm 1, line 13). The mutation applied in HC-GA is known in the literature of GAs as Bit-flip, and is widely used for binary encoded individuals. This operator consists of changing the binary genes according to a mutation rate *mr*, determining which instances will have, at a certain chosen gene, a "0" value changed to "1" and vice-versa [De Castro 2006]. Algorithm 3 shows the mutation process along with the related generalization and restriction operations.

Algorithm 3 Mutation and generalization/restriction operations.

```
Require: Population $P$, mutation rate $mr$, probability of using a test in a rule $pt$.
1: $mutationPop \leftarrow mutationSelection(P, mr)$
2: $remainingPop \leftarrow P - mutationPop$
3: for each $individual$ from $mutationPop$ do
4:     $number \leftarrow randomRealNumber(0, 1)$
5:     if $number <= 0.5$ then
6:         $individual \leftarrow flagMutation(individual, pt)$
7:     else
8:         for each $activeTest$ from $individual.activeTests$ do
9:             $operation \leftarrow randomIntNumber(0, 1)$
10:            if $operation = 0$ then
11:                $activeTest \leftarrow numericGeneralization(activeTest)$
12:            else if $operation = 1$ then
13:                $activeTest \leftarrow numericRestriction(activeTest)$
14:            end if
15:        end for
16:     end if
17: end for
18: $mutationPop \leftarrow mutationPop \cup remainingPop$
19: return $mutationPop$
```

4.6 Local Search

Both after a population is first generated (Algorithm 1, line 3) and when a new one is obtained by mutation (Algorithm 1, lines 13 and 14), these populations are submitted to a local search (Algorithm 1, lines 4 and 15). In HC-GA, a Min-Max Local Search is applied, as presented in Algorithm 4.

Algorithm 4 Min-Max local search.

```
Require: Population $P$, minimum number of covered instances per rule $minCov$, maximum number of covered
    instances per rule $maxCov$, maximum number of attempts to pass $maxAttempt$.
 1: for each $individual$ from $P$ do
 2:   $attempt \leftarrow 0$
 3:   $convergence \leftarrow 0$
 4:   while $convergence = 0$ and $attempt < maxAttempt$ do
 5:     $attempt \leftarrow attempt + 1$
 6:     if $coveredInstances < minCov$ then
 7:       if $|individual.activeTests| > 1$ then
 8:         $individual \leftarrow disableTestRandomly()$
 9:       else if $|individual.activeTests| = 1$ and $individual.activeTests[0].isNumeric()$ then
10:         $newTest \leftarrow generalizationOperator(individual.activeTests[0])$
11:         $individual \leftarrow updateTest(individual.activeTests[0], newTest)$
12:       end if
13:     else if $coveredInstances > maxCov$ then
14:       if $|individual.activeTests| < |datasetsAttributes|$ then
15:         $individual \leftarrow enableTestRandomly()$
16:       end if
17:     else
18:       $convergence \leftarrow 1$
19:     end if
20:   end while
21: end for
22: $newPopulation \leftarrow P$
23: return $newPopulation$
```

This local search aims to ensure that generated rules will cover between a minimum and a maximum of instances, being used to prevent the generation of rules which are too general or very specific, which could compromise the model accuracy and interpretability.

5. MATERIALS AND METHODS

This section presents the hierarchical datasets adopted along with their pre-processing, the literature ML classifiers implemented and moreover, the chosen hierarchical and conventional evaluation metrics.

5.1 Data Preparation

The two hierarchical datasets chosen are composed by TEs sequences collected from two public repositories, they are PGSB [1] and REPBASE [2]. The PGSB database [Nussbaumer et al. 2012] contains a compilation of plant repeat sequences, whilst REPBASE [Jurka et al. 2005] presents sequences of repetitive DNA from different eukaryotic species. Both repositories are reference databases being used in projects worldwide and do not contain the same sequences.

These datasets were organized according to a TEs hierarchical taxonomy proposed by [Wicker et al. 2007], which contains four hierarchical levels with many examples of TEs classes. Since this taxonomy is still under development and has been continuously adopted by the scientific community, it is expected that some TEs databases would not cover this entire taxonomy, as is the case for PGSB and REPBASE. Table I shows some statistics about these datasets. The number of features are the same for both bases which consists of k-mers quantified in the biological data with sizes of 2, 3 and 4. We chose K-mers is reason of being a popular feature used in Bioinformatics [Melsted and Pritchard

[1] http://pgsb.helmholtz-muenchen.de/plant/
[2] http://girinst.org/repbase/

2011]. The number of instances presented in the table column refers to the total number of instances, without division of train and test. For last, the remaining columns show the number of classes for each level of the TEs class hierarchy and the number of classes that are leaf node classes.

Table I. Datasets statistics.

Dataset	**Attributes**	**Instances**	**Classes per Level**	**Leaf Classes**
PGSB	336	18678	2 / 4 / 3 / 5	11
REPBASE	336	34559	2 / 5 / 12 / 9	24

5.2 Baseline Methods

Concerning the classifiers used in the experiments, our aim was to include at least one example of each classification paradigm, such as C4.5 (Symbolic), SVM (Probabilistic), KNN (Based on Examples) and the proposed method HC-GA, which follows an evolutionary approach. These methods were implemented using Python Scikit-learn [Pedregosa et al. 2011], a popular python-based library for ML, which its hyperparameters defined according to the official documentation [3]. Regarding the HC-GA, we built it from scratch using Java without any evolutionary framework. For last, the HC-GA's hyperparameters obtained by grid search and used in both datasets are shown in Table II.

Table II. HC-GA parameters.

Parameter	**Value**
Number of Generations	100
Population Size	100
Minimum No. of Covered Instances per Rule	10
Maximum No. of Covered Instances per Rule	6800 / 9000
Maximum No. of Uncovered Instances	10
Crossover Rate	90%
Mutation Rate	30%
Tournament Size	3
No. of Individuals Selected by Elitism	2
Probability of Using a Test in a Rule	3%

Even these final hyperparameters were obtained by fine-tuning, some of them were initially defined based on some assumptions that proved to be true after tuning. For example, a higher value for population size helped our method to find better rules, once more candidates for good rules were available. Concerning the high crossover rate, we realized this made it difficult to lose good rules and getting stuck in local optimal, since it makes the search more broad within the search space and, with combination of elitism, it also prevents the search from becoming random or going to bad regions. Also to avoid local optimal, we defined a low value for *minCov* and a high value for *maxCov* (see Algorithm 1), otherwise it would make the search too limited. It is appropriate to point out that *minCov* and *maxCov* are relative, defined according to the dataset used, in our case, *minCov* is roughly the number of instances for the minority class whereas *maxCov* is equal or less to the number of instances for the majority class. We used these criteria either to not losing rules that generalize well and to avoid limiting the search for good rules, since a short range of upper and lower bounds could lead to very specific rules with no interesting information about the data.

Regarding the tournament size and elitism rate, we chose to keep it small to avoid premature convergence and loss of variability. About the probability of using a test in a rule, a small value was

[3]http://scikit-learn.org/stable/supervised_learning

defined since many active tests would make a rule too specific or, otherwise, too general. This same logic was applied to the mutation rate. Thus, we defined a moderate value which made HC-GA found new good solutions in the decision space without being too sparse. Finally, the number of generations was set according to a gain versus runtime analysis.

5.3 Evaluation Metrics

As well as HC uses some data structures and specific approaches for training models, these problems require special evaluation metrics to proper analyze the performance of hierarchical classifiers [Costa et al. 2007]. This is due to the fact that traditional metrics employed in flat classification do not take into consideration the predictions at several levels of a hierarchy. Two popular metrics used in HC literature, coming from adaptations of traditional metrics, are the hierarchical Precision (hP) and hierarchical Recall (hR) [Kiritchenko et al. 2006]. Also, there is a metric that combines the previously mentioned, called hierarchical F-Measure (hF), which is the harmonic mean of hP and hR [Kiritchenko et al. 2006]. These metrics were presented in Equations 2, 3 and 4 in Section 5.3.

6. RESULTS AND DISCUSSION

This section presents experiments with the previously mentioned flat ML methods and HC-GA, executed in the same two TEs datasets, and which in order to present results with more statistical significance, were conducted using a 10-cross validation strategy. The analysis of the results is divided into HC experiments with some fitness implemented in HC-GA and the flat analysis comparing HC-GA with the other classifiers. The averages for the evaluation metrics adopted, along with their standard deviation values, are reported as follows.

6.1 Evaluation with Hierarchical Metrics

Since it is not possible to fully evaluate the performance of flat classifiers in a HC task, once they only make predictions in leaf levels (unless an adaption is made), results for the HC of TEs were obtained using only HC-GA. Table III shows the averages for the hierarchical F-Measure (hF) metric along with the respective standard deviation values. Its lines show the fitness functions employed and the columns present the results for each level of the TEs classes hierarchy, the leaf classes and the overall performance achieved by each function. It is worth noting that some values, as in *Level 3* and *Level 4*, appear as "-", which means that no prediction was made for that particular level.

Table III. Results using hierarchical metrics

	Level 1	**Level 2**	**Level 3**	**Level 4**	**Leaf Classes**	**Overall**
			PGSB			
hP	**0.95 ± 0.006**	**0.92 ± 0.009**	**0.78 ± 0.013**	**0.46 ± 0.055**	**0.84 ± 0.011**	**0.88 ± 0.008**
hR	0.94 ± 0.005	0.91 ± 0.007	0.77 ± 0.008	0.41 ± 0.047	0.83 ± 0.007	0.87 ± 0.005
hF	0.93 ± 0.005	0.91 ± 0.006	0.77 ± 0.011	0.43 ± 0.056	0.83 ± 0.009	0.87 ± 0.007
$AU(\overline{PRC})$	0.93 ± 0.005	0.90 ± 0.006	0.73 ± 0.012	0.24 ± 0.091	0.80 ± 0.008	0.85 ± 0.007
PC	0.54 ± 0.239	0.53 ± 0.230	0.08 ± 0.162	-	0.53 ± 0.085	0.43 ± 0.208
VG	0.90 ± 0.016	0.88 ± 0.011	0.63 ± 0.084	0.07 ± 0.043	0.74 ± 0.041	0.81 ± 0.016
			REPBASE			
hP	**0.90 ± 0.005**	**0.86 ± 0.004**	**0.68 ± 0.008**	0.35 ± 0.015	**0.71 ± 0.007**	**0.79 ± 0.004**
hR	0.89 ± 0.007	0.85 ± 0.007	**0.68 ± 0.007**	**0.36 ± 0.026**	0.69 ± 0.007	0.78 ± 0.006
hF	0.89 ± 0.007	0.85 ± 0.008	**0.68 ± 0.008**	**0.36 ± 0.019**	0.69 ± 0.007	0.78 ± 0.007
$AU(\overline{PRC})$	0.86 ± 0.012	0.77 ± 0.020	0.40 ± 0.098	0.24 ± 0.024	0.45 ± 0.075	0.68 ± 0.031
PC	0.27 ± 0.153	0.22 ± 0.137	-	-	0.17 ± 0.051	0.18 ± 0.109
VG	0.81 ± 0.045	0.74 ± 0.043	0.43 ± 0.083	0.01 ± 0.011	0.50 ± 0.073	0.65 ± 0.033

Basically, there are two groups of functions in these experiments, the one whose the main objective is to maximize the accuracy and the other that aims to achieve a better model interpretability. The first group is composed by hP, hR, hF and $AU(\overline{PRC})$, whereas PC and VG composes the other one. As shown in Table III, functions from the first group are very superior compared with the second, which in turn overcomes them in interpretability performance, as will be further discussed.

For both datasets, the hP function showed better or competitive results at all levels (except for *Level 4* in REPBASE), including the leaf node classes and the overall performance. However, functions hR and hF obtained very close results to that optimal in the four levels, being slightly superior to hP at *Level 4* in REPBASE, although, for that same level in PGSB, the differences were higher, so it is fair to consider that the performance of hP was superior in general. In addition, the last function of this previously called "first group", the $AU(\overline{PRC})$ function, has shown inferior results in comparison with others, presenting also higher standard deviation values. An explanation for the accuracy deterioration from a higher to a lower level, is that usually there are fewer instances at the lower levels of a hierarchy, which often hinders the classifier predictions. Considering the results presented in Table III, it is possible to conclude that some functions are more sensitive to the imbalance of instances at each level, as is the case of $AU(\overline{PRC})$.

Regarding the functions of the "second group", the PC and VG functions, were overcome by functions from the first group in all criteria evaluated and in both datasets. Although, these functions have contributed to interesting observations about the limitations of building interpretable models for HC problems. For example, considering the results obtained by PC, it is possible to note the difficult to made predictions at lower levels. Moreover, PC shows a high variation in performance when its standard deviation values are taken into account. In case of VG, the average results are better and closer to that obtained by the first group, showing also lower standard deviation, but still, higher values compared with others. We believe that more experiments could lead to improving the accuracy of such functions and even obtain models with a good trade-off between interpretability and accuracy.

Another analysis performed in this work is concerned with the interpretability of the models, shown in Table IV. Here we evaluate some features observed in the final models, such as the number of rules (*Rules*), number of active tests in a whole set of rules and the average of tests per rule (*Tests* and *perRule*), and the average of instances covered per rule (*Coverage*). The values presented below are the respective averages and standard deviation values for the functions implemented.

Table IV. Interpretability analysis

	PGSB			**REPBASE**		
	Rules	**Tests (per Rule)**	**Coverage**	**Rules**	**Tests (per Rule)**	**Coverage**
hP	1180 ± 15.47	8749 (7) ± 137.2	14 ± 0.316	2507 ± 27.88	19889 (7) ± 300.8	12 ± 0.000
hR	1154 ± 18.78	8376 (7) ± 117.3	14 ± 0.000	2424 ± 22.72	18914 (7) ± 165.0	12 ± 0.316
hF	1156 ± 18.12	8366 (7) ± 0.000	14 ± 0.000	2411 ± 34.80	18676 (7) ± 315.0	12 ± 0.422
$AU(\overline{PRC})$	367 ± 13.63	2257 (6) ± 108.0	45 ± 1.829	204 ± 43.91	1251 (6) ± 306.2	162 ± 53.06
PC	**4 ± 0.699**	**15 (4) ± 2.898**	**4819 ± 863.8**	**4 ± 0.000**	**23 (6) ± 4.442**	**7775 ± 2.214**
VG	23 ± 6.683	117 (5) ± 29.05	802 ± 286.1	27 ± 8.044	143 (5) ± 52.95	1275 ± 405.6

As expected, functions VG and PC obtained the best interpretable models, the latter being the best one in the whole experiments performed. An important observation about this analysis together with that presented in Table III is that function VG is the most balanced function with respect to the accuracy and interpretability performances. In relation to the functions that obtained the best hF values in the previous analysis, here they had the worst performances regarding model interpretability, being much inferior to the others verified. This contributes to the affirmation of how difficult is to obtain interpretable models with good accuracy, and how these are contradictory objectives.

6.2 Evaluation with Non-Hierarchical Metrics

Table V shows the results for the flat classification task obtained by HC-GA and the flat classifiers, evaluated according to Precision (P), Recall (R) and F1 score (F). The HC-GA fitness used here is

hP, given its best results compared to other functions. These experiments were performed in order to answer two research questions: (i) How good is HC-GA in predicting the leaf node classes of TEs; and (ii) How are flat classifiers able to make predictions for hierarchical datasets and which one presents the highest accuracy in predicting the leaf classes of TEs.

Table V. Results using non-hierarchical metrics

	C4.5	**NB**	**KNN**	**SVM**	**MLP**	**HC-GA**
				PGSB		
P	0.66 ± 0.011	0.45 ± 0.041	0.80 ± 0.009	0.68 ± 0.022	0.41 ± 0.141	**0.84 ± 0.011**
R	0.66 ± 0.011	0.20 ± 0.009	0.80 ± 0.009	0.69 ± 0.012	0.30 ± 0.110	**0.85 ± 0.014**
F	0.66 ± 0.011	0.28 ± 0.016	0.80 ± 0.009	0.68 ± 0.017	0.32 ± 0.072	**0.84 ± 0.011**
				REPBASE		
P	0.46 ± 0.008	0.38 ± 0.111	0.69 ± 0.006	0.56 ± 0.013	0.24 ± 0.086	**0.72 ± 0,007**
R	0.46 ± 0.007	0.11 ± 0.003	0.68 ± 0.007	0.49 ± 0.004	0.22 ± 0.056	**0.70 ± 0.010**
F	0.46 ± 0.007	0.16 ± 0.014	0.69 ± 0.006	0.52 ± 0.008	0.22 ± 0.051	**0.71 ± 0.007**

As result, KNN obtained the best averages including low standard deviations among the flat classifiers, being superior to others in the three metrics and in both datasets. Although, HC-GA presented the best results among all ML classifiers, even that was not been designed for this purpose. In addition, it is worth noting that the performance of all methods is enough decreased in REPBASE when compared with the one obtained in PGSB. We believe this was caused in reason of the imbalanced nature of these datasets, which is even more present in REPBASE. Because of this particularity, we can conclude that this may lead to the bad performance of the flat classifiers, that are even more sensitive to imbalanced classes. In general, flat classifiers do not perform well predicting classes for hierarchical datasets, even ignoring hierarchical relationships among classes.

7. CONCLUSIONS AND FUTURE RESEARCH

This work presented an extension of the original paper where HC-GA, a global hierarchical method capable of generating HC rules with good accuracy and interpretability, was proposed. As in the previous study, the HC-GA was compared with some traditional flat classifiers in the prediction of TEs leaf classes. As seen in the Results and Discussion section, the improved HC-GA overcame the flat methods' prediction of the leaf node classes in both datasets for the three evaluation metrics applied. In a HC problem, the task of predicting classes at multiples levels of a hierarchy presents an increasing difficulty as this hierarchy is explored towards the leaves, since it is common that fewer examples of leaf classes are available, in contrast to the greater variety of examples at higher levels. This particularity of HC problems usually makes classifiers present poor performance in the lower levels, which was the case for flat classifiers, even though they ignore the relationships among classes. We performed some experiments with a variety of new fitness functions with the objective of verifying the quality of the HC-GA predictions not only in general but also in all levels including leaf node classes. We believe that our method presented a good variety of functions that are alternatives with different accuracy and interpretability performances to be applied for other HC problems in many domains. For future research we intend to continue with the experiments of different functions in HC-GA, making combinations of functions aiming to explore their different aspects, such as those of the two groups that we presented. Therefore, the next step in the research is to perform experiments with Multi-Objective GAs. Also, we plan to compare the global method HC-GA with other HC methods, both global and local, to validate more properly the results obtained. Besides, more analysis aiming to investigate the weaknesses of HC-GA will be conducted, such as collecting and comparing the runtimes required for each fitness.

REFERENCES

Abrusán, G., Grundmann, N., DeMester, L., and Makalowski, W. Teclass—a tool for automated classification of unknown eukaryotic transposable elements. *Bioinformatics* 25 (10): 1329–1330, 2009.

Altschul, S. F., Gish, W., Miller, W., Myers, E. W., and Lipman, D. J. Basic local alignment search tool. *Journal of molecular biology* 215 (3): 403–410, 1990.

Biémont, C. A brief history of the status of transposable elements: from junk dna to major players in evolution. *Genetics* 186 (4): 1085–1093, 2010.

Costa, E. P., Lorena, A. C., Carvalho, A. C. P. L. F., and Freitas, A. A. Comparing several approaches for hierarchical classification of proteins with decision trees. In *II Brazilian Symposium on Bioinformatics.* Lecture Notes in Bioinformatics, vol. 4643. Springer-Verlag, Berlin, Heidelberg, pp. 126–137, 2007.

Davis, J. and Goadrich, M. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning.* ACM, pp. 233–240, 2006.

De Castro, L. N. *Fundamentals of natural computing: basic concepts, algorithms, and applications.* CRC Press, 2006.

Feschotte, C. Transposable elements and the evolution of regulatory networks. *Nature Reviews Genetics* 9 (5): 397–405, 2008.

Feschotte, C., Keswani, U., Ranganathan, N., Guibotsy, M. L., and Levine, D. Exploring repetitive DNA landscapes using REPCLASS, a tool that automates the classification of transposable elements in eukaryotic genomes. *Genome biology and evolution* vol. 1, pp. 205–220, 2009.

Ghazi, D., Inkpen, D., and Szpakowicz, S. Hierarchical versus flat classification of emotions in text. In *Proceedings of the NAACL HLT 2010 workshop on computational approaches to analysis and generation of emotion in text.* Association for Computational Linguistics, pp. 140–146, 2010.

Hoede, C., Arnoux, S., Moisset, M., Chaumier, T., Inizan, O., Jamilloux, V., and Quesneville, H. Pastec: an automatic transposable element classification tool. *PLoS One* 9 (5): e91929, 2014.

Holland, J. H. Escaping brittleness: The possibilities of general-purpose learnlng algorithms applied to parallel rule-based systems. *Machine learning: An artificial intelligence approach* vol. 2, 1986.

Jurka, J., Kapitonov, V. V., Pavlicek, A., Klonowski, P., Kohany, O., and Walichiewicz, J. Repbase update, a database of eukaryotic repetitive elements. *Cytogenetic and genome research* 110 (1-4): 462–467, 2005.

Kazazian, H. H. Mobile elements: drivers of genome evolution. *science* 303 (5664): 1626–1632, 2004.

Kiritchenko, S., Matwin, S., and Famili, A. F. Functional annotation of genes using hierarchical text categorization. In *in Proc. of the BioLINK SIG: Linking Literature, Information and Knowledge for Biology (held at ISMB-05.* Citeseer, Uottawa CA, 2005.

Kiritchenko, S., Matwin, S., Nock, R., and Famili, A. F. Learning and evaluation in the presence of class hierarchies: Application to text categorization. In *Conference of the Canadian Society for Computational Studies of Intelligence.* Springer, Springer Berlin Heidelberg, pp. 395–406, 2006.

Loureiro, T., Camacho, R., Vieira, J., and Fonseca, N. A. Boosting the Detection of Transposable Elements Using Machine Learning. *6th International Conference on Practical Applications of Computational Biology & Bioinformatics* 154 (222): 85–91, 2013.

McClintock, B. The significance of responses of the genome to challenge, 1993.

Melsted, P. and Pritchard, J. K. Efficient counting of k-mers in dna sequences using a bloom filter. *BMC bioinformatics* 12 (1): 333, 2011.

Mitchell, T. M. et al. Machine learning, 1997.

Nussbaumer, T., Martis, M. M., Roessner, S. K., Pfeifer, M., Bader, K. C., Sharma, S., Gundlach, H., and Spannagl, M. Mips plantsdb: a database framework for comparative plant genome research. *Nucleic acids research* 41 (D1): D1144–D1151, 2012.

Otero, F. E., Freitas, A. A., and Johnson, C. G. A new sequential covering strategy for inducing classification rules with ant colony algorithms. *IEEE Transactions on Evolutionary Computation* 17 (1): 64–76, 2013.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* vol. 12, pp. 2825–2830, 2011.

Pereira, G. T. and Cerri, R. Classificação hierárquica e não hierárquica de elementos transponíveis. *Proceedings of the 5th Symposium on Knowledge Discovery, Mining and Learning (KDMiLe)*, 2017.

Rebollo, R., Romanish, M. T., and Mager, D. L. Transposable elements: an abundant and natural source of regulatory sequences for host genes. *Annual review of genetics* vol. 46, pp. 21–42, 2012.

Silla, C. N. and Kaestner, C. A. Hierarchical classification of bird species using their audio recorded songs. In *Systems, Man, and Cybernetics (SMC), 2013 IEEE International Conference on.* IEEE, pp. 1895–1900, 2013.

Silla, CarlosN., J. and Freitas, A. A survey of hierarchical classification across different application domains. *Data Mining and Knowledge Discovery* 22 (1-2): 31–72, 2011.

Smit, A. F., Hubley, R., and Green, P. RepeatMasker, 1996.

Smith, S. F. *A learning system based on genetic adaptive algorithms*. Ph.D. thesis, University of Pittsburgh, Pittsburgh, PA, 1980.

Srinivasan, S. and Ramakrishnan, S. Evolutionary multi objective optimization for rule mining: a review. *Artificial Intelligence Review* 36 (3): 205, 2011.

Steinbiss, S., Willhoeft, U., Gremme, G., and Kurtz, S. Fine-grained annotation and classification of de novo predicted ltr retrotransposons. *Nucleic acids research* 37 (21): 7002–7013, 2009.

van de Lagemaat, L. N., Landry, J.-R., Mager, D. L., and Medstrand, P. Transposable elements in mammals promote regulatory variation and diversification of genes with specialized functions. *Trends in Genetics* 19 (10): 530–536, 2003.

Vens, C., Struyf, J., Schietgat, L., Džeroski, S., and Blockeel, H. Decision trees for hierarchical multi-label classification. *Machine Learning* 73 (2): 185–214, 2008.

Wetzel, A. *Evaluation of the effectiveness of genetic algorithms in combinatorial optimization*. Ph.D. thesis, University of Pittsburgh, Pittsburgh, PA, 1983.

Wheeler, T. J., Clements, J., Eddy, S. R., Hubley, R., Jones, T. A., Jurka, J., Smit, A. F., and Finn, R. D. Dfam: a database of repetitive dna based on profile hidden markov models. *Nucleic acids research* 41 (D1): D70–D82, 2012.

Wicker, T., Sabot, F., Hua-Van, A., Bennetzen, J. L., Capy, P., Chalhoub, B., Flavell, A., Leroy, P., Morgante, M., Panaud, O., et al. A unified classification system for eukaryotic transposable elements. *Nature Reviews Genetics* 8 (12): 973–982, 2007.

Zimek, A., Buchwald, F., Frank, E., and Kramer, S. A study of hierarchical and flat classification of proteins. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 7 (3): 563–571, 2010.