Optimum-Path Forest Pruning Parameter Estimation Through Harmony Search

Rodrigo Yuji Mizobe Nakamura, Clayton Reginaldo Pereira, João Paulo Papa
Department of Computing
UNESP - Univ Estadual Paulista
Bauru, São Paulo
Email: rodrigo.mizobe@fc.unesp.br

Alexandre Xavier Falcão
Institute of Computing
University of Campinas
Campinas, São Paulo
Email: afalcao@ic.unicamp.br

***Abstract*—Pattern recognition in large amount of data has been paramount in the last decade, since that is not straightforward to design interactive and real time classification systems. Very recently, the Optimum-Path Forest classifier was proposed to overcome such limitations, together with its training set pruning algorithm, which requires a parameter that has been empirically set up to date. In this paper, we propose a Harmony Search-based algorithm that can find near optimal values for that. The experimental results have showed that our algorithm is able to find proper values for the OPF pruning algorithm parameter.**

***Keywords*-Optimum-Path Forest; Supervised classification; Pattern Recognition;**

I. Introduction

The problem of large datasets classification still remains pursued by scientific community. Images acquired by personal digital cameras have millions of pixels to be recognized, for instance. Genre music classification of high quality audio files in multimedia collections is another critical problem that require fast and reliable recommendation systems. Magnetic resonance images from human brain produce hundreds of thousands of voxels for further classification and identification of possible diseases. Anyway, one has several problems in which a prompt feedback is more important than an accurate, but expensive, classifier's decision.

In order to handle such challenge, Papa et al. [1] have proposed a new way of facing pattern recognition by modeling it as a graph partition in optimum-path trees (OPTs), which are rooted at some key samples (prototypes). The main idea is to begin a competition process among prototypes in order to conquer the remaining dataset samples. This process is ruled by a smooth path-cost function defined by the user, which stands for the optimality criterion. The collection of OPTs defines an optimum-path forest (OPF), which gives the name to the classifier. The OPF has demonstrated to be similar to Support Vector Machines (SVMs) [2], but much faster for training. This skill may be very important in the aforementioned context, i.e., the one that stands for applications that are characterized by a large amount of data. Therefore, it is often desirable to have user-friendly tools that can interact with fast feedback.

Further, aiming to speed up OPF classification phase, Papa et al. [3] proposed a training set pruning algorithm, which can design compact and representative training sets. The idea is to use an evaluating set to identify the irrelevant training samples, i.e., the ones that did not participate in any classification process over the evaluating set. Then, the algorithm removes those samples from training set. This procedure has demonstrated to be interesting in situations in which we have redundant samples, such as image classification and network intrusion detection benchmarking datasets. In the former case, one may have similar features for pixels in the same neighborhood, and in the latter situation several connections in a short period of time may define similar samples in the feature space.

The OPF training set pruning algorithm works with a stopping criterion, which is called $MLoss$ and is defined as the absolute deviation between the accuracy over the original evaluating set and the final pruned one. Although this approach has obtained good results in several applications [4], [5], it is not straightforward to find a good value for $MLoss$, which has been performed empirically up to date. In this paper, we propose an algorithm to find proper values for that using an evolutionary intelligence algorithm called Harmony Search (HS) [6]. Experimental results have showed that the proposed approach is suitable to this task. The remainder of this paper is organized as follows. Sections II and III review OPF and HS background, respectively. Section IV presents the proposed methodology to estimate $MLoss$ and Section V refers to the experimental section. Finally, Section VI states conclusions.

II. Optimum-Path Forest

The OPF classifier works by modeling the problem of pattern recognition as a graph partition in a given feature space. The nodes are represented by the feature vectors and the edges connect all pairs of them, defining a full connectedness graph. This kind of representation is straightforward, given that the graph does not need to be explicitly represented, allowing us to save memory. The partition of the graph is carried out by a competition process between some key samples (*prototypes*), which offer optimum paths to the remaining nodes of the graph. Each prototype sample defines its optimum-path tree (OPT), and the collection of all OPTs defines an optimum-path forest, which gives the name to the classifier [1]. The OPF can be seen as a generalization of the well known Dijkstra's algorithm to compute optimum paths from a source node to

the remaining ones [7]. The main difference relies on the fact that OPF uses a set of source nodes (prototypes) with any smooth path-cost function [8].

A. Background theory

Let $Z = Z_1 \cup Z_2 \cup Z_3$ be a dataset labeled with a function λ, in which Z_1, Z_2 and Z_3 are, respectively, training, evaluating and test sets such that Z_1 and Z_2 are used to design a given classifier and Z_3 is used to assess its accuracy. Let $S \subseteq Z_1$ a set of prototype samples. Essentially, the OPF classifier creates a discrete optimal partition of the feature space such that any sample $s \in Z_2 \cup Z_3$ can be classified according to this partition. This partition is an optimum path forest (OPF) computed in $\Re^n$ by the image foresting transform (IFT) algorithm [8].

The OPF algorithm may be used with any *smooth* path-cost function which can group samples with similar properties [8]. Particularly, we used the path-cost function f_{max}, which is computed as follows:

$$\begin{aligned} f_{max}(\langle s \rangle) &= \begin{cases} 0 & \text{if } s \in S, \\ +\infty & \text{otherwise} \end{cases} \\ f_{max}(\pi \cdot \langle s,t \rangle) &= \max\{f_{max}(\pi), d(s,t)\}, \end{aligned} \quad (1)$$

in which $d(s,t)$ means the distance between samples s and t, and a path π is defined as a sequence of adjacent samples. In such a way, we have that $f_{max}(\pi)$ computes the maximum distance between adjacent samples in π, when π is not a trivial path.

The OPF algorithm assigns one optimum path $P^*(s)$ from S to every sample $s \in Z_1$, forming an optimum path forest P (a function with no cycles which assigns to each $s \in Z_1 \backslash S$ its predecessor $P(s)$ in $P^*(s)$ or a marker nil when $s \in S$. Let $R(s) \in S$ be the root of $P^*(s)$ which can be reached from $P(s)$. The OPF algorithm computes for each $s \in Z_1$, the cost $C(s)$ of $P^*(s)$, the label $L(s) = \lambda(R(s))$, and the predecessor $P(s)$.

The OPF classifier is composed of two distinct phases: (i) training and (ii) classification. The former step consists, essentially, in finding the prototypes and computing the optimum-path forest, which is the union of all OPTs rooted at each prototype. After that, we take a sample from the test sample, connect it to all samples of the optimum-path forest generated in the training phase and we evaluate which node offered the optimum path to it. Notice that this test sample is not permanently added to the training set, i.e., it is used only once. The next sections describe in details this procedure.

1) Training: We say that S^* is an optimum set of prototypes when the OPF algorithm minimizes the classification errors for every $s \in Z_1$. S^* can be found by exploiting the theoretical relation between minimum-spanning tree (MST) and optimum-path tree for f_{max} [9]. The training essentially consists in finding S^* and an OPF classifier rooted at S^*.

By computing an MST in the complete graph (Z_1, A), we obtain a connected acyclic graph whose nodes are all samples of Z_1 and the arcs are undirected and weighted by the distances d between adjacent samples. The spanning tree is optimum in the sense that the sum of its arc weights is minimum as compared to any other spanning tree in the complete graph. In the MST, every pair of samples is connected by a single path which is optimum according to f_{max}. That is, the minimum-spanning tree contains one optimum-path tree for any selected root node. The optimum prototypes are the closest elements of the MST with different labels in Z_1 (i.e., elements that fall in the frontier of the classes). *Algorithm 1* implements the training procedure for OPF.

Algorithm 1: – OPF TRAINING ALGORITHM

INPUT: *A λ-labeled training set Z_1 and the pair (v,d) for feature vector and distance computations.*
OUTPUT: *Optimum-path forest P_1, cost map C_1, label map L_1, and ordered set Z_1'.*
AUXILIARY: *Priority queue Q, set S of prototypes, and cost variable cst.*

1. Set $Z_1' \leftarrow \emptyset$ and compute by MST the prototype set $S \subset Z_1$.
2. For each $s \in Z_1 \backslash S$, set $C_1(s) \leftarrow +\infty$.
3. For each $s \in S$, do
4. $\quad C_1(s) \leftarrow 0$, $P_1(s) \leftarrow nil$, $L_1(s) \leftarrow \lambda(s)$, insert s in Q.
5. While Q is not empty, do
6. $\quad$ Remove from Q a sample s such that $C_1(s)$ is minimum.
7. $\quad$ Insert s in Z_1'.
8. $\quad$ For each $t \in Z_1$ such that $C_1(t) > C_1(s)$, do
9. $\quad\quad$ Compute $cst \leftarrow \max\{C_1(s), d(s,t)\}$.
10. $\quad\quad$ If $cst < C_1(t)$, then
11. $\quad\quad\quad$ If $C_1(t) \neq +\infty$, then remove t from Q.
12. $\quad\quad\quad$ $P_1(t) \leftarrow s$, $L_1(t) \leftarrow L_1(s)$, $C_1(t) \leftarrow cst$.
13. $\quad\quad\quad$ Insert t in Q.
14. Return a classifier $[P_1, C_1, L_1, Z_1']$.

The time complexity for training is $\theta(|Z_1|^2)$, due to the main (Lines 5-13) and inner loops (Lines 8-13) in *Algorithm 1*, which run $\theta(|Z_1|)$ times each.

2) Classification: For any sample $t \in Z_3$ (similar definition is applied to Z_2), we consider all arcs connecting t with samples $s \in Z_1$, as though t were part of the training graph. Considering all possible paths from S^* to t, we find the optimum path $P^*(t)$ from S^* and label t with the class $\lambda(R(t))$ of its most strongly connected prototype $R(t) \in S^*$. This path can be identified incrementally by evaluating the optimum cost $C(t)$ as

$$C(t) = \min\{\max\{C(s), d(s,t)\}\}, \ \forall s \in Z_1. \quad (2)$$

Let the node $s^* \in Z_1$ be the one that satisfies Equation 2 (i.e., the predecessor $P(t)$ in the optimum path $P^*(t)$). Given that $L(s^*) = \lambda(R(t))$, the classification simply assigns $L(s^*)$ as the class of t. An error occurs when $L(s^*) \neq \lambda(t)$. *Algorithm 2* implements this procedure.

Algorithm 2: – OPF CLASSIFICATION ALGORITHM

INPUT: *Classifier $[P_1, C_1, L_1, Z_1']$, test set Z_3, and the pair (v,d) for feature vector and distance computations.*
OUTPUT: *Label L_2 and predecessor P_2 maps defined for Z_3, and accuracy value Acc.*
AUXILIARY: *Cost variables tmp and $mincost$.*

1. For each $t \in Z_3$, do
2. $\quad$ $i \leftarrow 1$, $mincost \leftarrow max\{C_1(k_i), d(k_i,t)\}$.

3. $L_2(t) \leftarrow L_1(k_i)$ and $P_2(t) \leftarrow k_i$.
4. While $i < |Z_1'|$ and $mincost > C_1(k_{i+1})$, do
5. Compute $tmp \leftarrow \max\{C_1(k_{i+1}, d(k_{i+1}, t)\}$.
6. If $tmp < mincost$, then
7. $mincost \leftarrow tmp$.
8. $L_2(t) \leftarrow L(k_{i+1})$ and $P_2(t) \leftarrow k_{i+1}$.
9. $i \leftarrow i + 1$.
10. Compute accuracy Acc according to [1].
11. Return $[L_2, P_2, Acc]$.

In *Algorithm 2*, the main loop (Lines $1-9$) performs the classification of all nodes in Z_3. The inner loop (Lines $4-9$) visits each node $k_{i+1} \in Z_1'$, $i = 1, 2, \ldots, |Z_1'| - 1$ until an optimum path $\pi_{k_{i+1}} \cdot \langle k_{i+1}, t \rangle$ is found.

3) Learning and pruning: Large datasets usually present redundancy, so at least in theory it should be possible to estimate a reduced training set with the most relevant patterns for classification. The use of a training set Z_1 and an evaluation set Z_2 has allowed OPF to learn relevant samples for Z_1 from the classification errors in Z_2, by swapping misclassified samples of Z_2 and non-prototype samples of Z_1 during a few iterations [1]. In this learning strategy, Z_1 remains the same size and the classifier instance with the highest accuracy is selected to be tested on the unseen set Z_3. Algorithm 3 implements this learning procedure.

Algorithm 3: – OPF LEARNING ALGORITHM

INPUT: *A λ-labeled training and evaluating sets Z_1 and Z_2, respectively, number T of iterations, and the pair (v, d) for feature vector and distance computations.*
OUTPUT: *Optimum-path forest P_1, cost map C_1, label map L_1, ordered set Z_1' and $MaxAcc$.*
AUXILIARY: *Arrays FP and FN of sizes c for false positives and false negatives, set S of prototypes, and list LM of misclassified samples.*

1. Set $MaxAcc \leftarrow -1$.
2. For each iteration $I = 1, 2, \ldots, T$, do
3. $LM \leftarrow \emptyset$ and compute the set $S \subset Z_1$ of prototypes.
4. $[P_1, C_1, L_1, Z_1'] \leftarrow$ Algorithm $1(Z_1, S, (v, d))$.
5. For each class $i = 1, 2, \ldots, c$, do
6. $FP(i) \leftarrow 0$ and $FN(i) \leftarrow 0$.
7. $[L_2, P_2, Acc] \leftarrow$ Algorithm $2([P_1, C_1, L_1, Z_1'], Z_2, (v, d))$
8. If $Acc > MaxAcc$ then
9. Save the current instance $[P_1, C_1, L_1, Z_1']$
10. of the classifier and set $MaxAcc \leftarrow Acc$.
11. While $LM \neq \emptyset$
12. $LM \leftarrow LM \backslash t$.
13. Replace t by a non-prototype sample, randomly
14. selected from Z_1.
15. Return the classifier instance $[P_1, C_1, L_1, Z_1']$ with the highest
16. accuracy in Z_2, and its value $MaxAcc$.

The efficacy of *Algorithm 3* increases with the size of Z_1, because more non-prototype samples can be swapped by misclassified samples of Z_2. However, for sake of efficiency, we need to choose a reasonable maximum size for Z_1. After learning the best training samples for Z_1, we may also mark paths in P_1 used to classify samples in Z_2 and define their nodes as *relevant samples* in a set $\mathcal{R}$. The "irrelevant" training samples in $Z_1 \backslash \mathcal{R}$ can then be moved to Z_2. *Algorithm 4* applies this idea repetitively, while the loss in accuracy on Z_2 with respect to the highest accuracy obtained by *Algorithm 3* (using the initial training set size) is less or equal to a maximum value $MLoss$ specified by the user or there are no more irrelevant samples in Z_1.

Algorithm 4: – LEARNING-WITH-PRUNING ALGORITHM

INPUT: *Training and evaluation sets, Z_1 and Z_2, labeled by λ, the pair (v, d) for feature vector and distance computations, maximum loss $MLoss$ in accuracy on Z_2, and number T of iterations.*
OUTPUT: *EOPF classifier $[P_1, C_1, L_1, Z_1']$ with reduced training set.*
AUXILIARY: *Set $\mathcal{R}$ of relevant samples, and variables Acc and tmp.*

1. $[P_1, C_1, L_1, Z_1'] \leftarrow$ Algorithm $3(Z_1, Z_2, T, (v, d))$.
2. $[L_2, P_2, Acc] \leftarrow$ Algorithm $2([P_1, C_1, L_1, Z_1'], Z_2, (v, d))$.
3. $tmp \leftarrow Acc$ and $\mathcal{R} \leftarrow \emptyset$.
4. While $|Acc - tmp| \leq MLoss$ and $\mathcal{R} \neq Z_1$ do
5. $\mathcal{R} \leftarrow \emptyset$.
6. For each sample $t \in Z_2$, do
7. $s \leftarrow P_2(t) \in Z_1$.
8. While $s \neq nil$, do
9. $\mathcal{R} \leftarrow \mathcal{R} \cup s$.
10. $s \leftarrow P_1(s)$.
11. Move samples from $Z_1 \backslash \mathcal{R}$ to Z_2.
12. $[P_1, C_1, L_1, Z_1'] \leftarrow$ Algorithm $3(Z_1, Z_2, T, (v, d))$.
13. $[L_2, P_2, tmp] \leftarrow$ Algorithm $2([P_1, C_1, L_1, Z_1'], Z_2, (v, d))$.
14. Return $[P_1, C_1, L_1, Z_1']$.

In *Algorithm 4*, Lines $1-3$ compute learning and classification using the highest accuracy classifier obtained for an initial training set size. Its accuracy is returned in Acc and used as reference value in order to stop the pruning process, when the loss in accuracy is greater than a user-specified $MLoss$ value or when all training samples are considered relevant. The main loop in Lines $4-13$ essentially marks the relevant samples in Z_1 by following the optimum paths used for classification (Lines $5-10$) backwards, moves irrelevant samples to Z_2, and repeats learning and classification from a reduced training set until it reaches the above stopping criterion.

III. HARMONY SEARCH

The Harmony Search (HS) is an evolutionary algorithm inspired in the music, considering the improvisation process of music players [6]. The HS is simple in concept, few in parameters, and easy in implementation, with theoretical background of stochastic derivative. The main idea is to use the same process adopted by musicians to create new songs to obtain a near-optimal solution of some optimization process. Basically, any possible solution is modeled as a harmony and each parameter to be optimized can be seen as a musical note. The best harmony (solution) is chosen as the one that maximizes some optimization criterion. The algorithm is composed by few steps, as described below:

- Step 1: Initialize the optimization problem and algorithm parameters;

- Step 2: Initialize a Harmony Memory (HM);
- Step 3: Improvise a new harmony from HM;
- Step 4: Update the HM if the new harmony is better than the worst harmony in the HM. In this case, one includes the new harmony in HM, and removes the worst one from HM; and
- Step 5: If the stopping criterion is not satisfied, go to Step 3.

Follow, we discuss each one of the aforementioned steps.

a) The Optimization Problem and Algorithm Parameters: In order to describe how HS works, an optimization problem is specified in Step 1 as follows:

$$Minimize\ f(x)\ subject\ to\ x_i,\ \forall i = 1, 2, \ldots, HMS, \quad (3)$$

where $f(x)$ is the objective function, x_i means the harmony i and HMS is the number of harmonies (harmony memory size - HMS).

The HS algorithm parameters required to solve the optimization problem (Equation 3) are also specified in this step: HMS, harmony memory considering rate (HMCR), pitch adjusting rate (PAR), and stopping criterion. HMCR and PAR are parameters used to improve the solution vector, i.e., they can help the algorithm to find globally and locally improved solutions in the harmony search process (Step 3).

b) Harmony Memory (HM): Now, let us define x_i^j as the j-th value of harmony i. In Step 2, the HM matrix (Equation 4) is initialized with randomly generated solution vectors with their respective values of the objective function:

$$HM = \left[\begin{array}{cccc|c} x_1^1 & x_1^2 & \ldots & x_1^n & f(x_1) \\ x_2^1 & x_2^2 & \ldots & x_2^n & f(x_2) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{hms}^1 & x_{hms}^2 & \ldots & x_{hms}^n & f(x_{hms}) \end{array}\right]. \quad (4)$$

c) Generating a New Harmony From HM: In Step 3, a new harmony vector $x_i^{'}$ is generated from the HM based on memory considerations, pitch adjustments, and randomization (music improvisation). It is also possible to choose the new value using the HMCR parameter, which varies between 0 and 1 as follows:

$$x_i' \leftarrow \begin{cases} x_i' \in \{x_i^1, x_i^2, \ldots, x_i^{HMS}\} & \text{with probability HCMR,} \\ x_i' \notin X_i & \text{with probability (1-HCMR).} \end{cases} \quad (5)$$

The HMCR is the probability of choosing one value from the historic values stored in the HM, and (1- HMCR) is the probability of randomly choosing one feasible value not limited to those stored in the HM.

Further, every component of the new harmony vector $x_i^{'}$ is examined to determine whether it should be pitch-adjusted:

$$\text{Pitching adjusting decision for } x_i' \leftarrow \begin{cases} \textbf{Yes} \text{ with probability PAR,} \\ \textbf{No} \text{ with probability (1-PAR).} \end{cases} \quad (6)$$

The pitch adjustment of each instrument is often used to improve the solutions and to escape from local optima. This mechanism concerns with shifting the neighboring values of some decision variable in the harmony. If the pitch adjustment decision for the decision variable x_1' is Yes, x_1' is replaced as follows:

$$x_i' \leftarrow x_i' + \sigma, \quad (7)$$

where σ is an arbitrary value.

d) Update HM: In Step 4, if the new harmony vector is better than the worst harmony in the HM, the latter is replaced by this new harmony.

e) Stopping Criterion: In Step 5, if the HS algorithm finishes when it satisfies the stopping criterion. Otherwise, Steps 3 and 4 are repeated in order to improvise a new harmony again.

IV. Proposed method

In this section, we present our proposed algorithm to find the $MLoss$ value automatically, which falls in the finite interval $[0, 100]$. The main idea is to use HS to find proper values for that, since we have an infinite number of real valued possible solutions, and an exhaustive search may be prohibitive.

As any optimization algorithm, HS also needs a fitness function to maximize (minimize). In our schema, it is desirable to find the best trade-off between accuracy over the evaluating set and the training set size. Therefore, we would like to find values for $MLoss$ such that the accuracy can the best as possible, and the training set size can be the small as we can obtain. Thus, in order to deal with that, we proposed the following fitness function:

$$F = Acc\left(\frac{1}{1 + e^{-|Z_1|}}\right)^{-1}, \quad (8)$$

in which Acc stands for the accuracy over the evaluating set. The idea for that is to make Acc and the training set size inversely proportional to each other, in the sense that whether the accuracy stabilizes, the fitness function can still increases with lower values for $|Z_1|$. However, the reader may ask about a simple rate between accuracy and training set size. Although our first experiments were conducted with that idea, the training set size dominates the accuracy value, leading us to high pruning rates and low accuracy ones.

Thus, in order to smooth the dominance of training set size, we decided to use a sigmoid function, since the second term of Equation 8, i.e., $\frac{1}{1+e^{-|Z_1|}}$ falls in the interval $[0.5, 1]$, given that $0 < |Z_1| < +\infty$. *Algorithm 5* implements our proposed methodology to find $MLoss$.

The initial loop in Lines $1-5$ is responsible for the harmony memory initialization. In Line 2, each harmony receives a real value randomly selected within the interval $[0, 100]$. Further, *Algorithm* 4 is called with $MLoss = HM_{i,1}$ in Line 3. After that, the classifier generated in Line 3 is used to access the accuracy over the evaluating set Z_2 (Line 4). The accuracy

value is stored in $HM_{i,2}$ position, which stands for $f(x_i)$ (Equation 4).

The conditional in Line 7 verifies whether the new harmony h will be composed by some value extracted from the harmony memory (Lines $8-9$). In the affirmative case, Line 8 selects an index value within $[1, hms]$ in order to retrieve the $MLoss$ value from some position of the harmony memory. Otherwise, the new harmony will be composed by a randomly number within the interval $[0, 100]$ in Line 12 (these steps implement Equation 5). Lines $10-11$ implements the pitch adjusting rate, as described in Equation 7. As the value of $MLoss$ can not change a lot, we used $\sigma = 0.1$. The Lines $13-15$ calculate the fitness value of the new harmony h, which is stored in $h[2]$ ($h[1]$ stores the $MLoss$ value).

Line 16 finds the harmony that leads to the lowest fitness in the harmony memory, and stores its index in the i_{min} variable. The corresponding fitness is stored in $minFitness$. The next step concerns with replacing the worst harmony with the new one obtained in the previous step whether its fitness value is lower than the one provided by the new harmony. Lines $17-19$ implements this idea. The function call $GetMaxHarmony$ in Line 20 finds the harmony i that achieved the highest fitness, and store its optimized value in Line 20. Finally, Line 21 outputs $MLoss$.

Algorithm 5: – PROPOSED ALGORITHM

INPUT: *Labeled training Z_1 and evaluating set Z_2, the pair (v, d) for feature vector and distance computations, harmony memory size hms, harmony memory considering rate $hmcr$, pitch adjusting rate par and number of iterations T for convergence.*

OUTPUT: *$MLoss$ value.*

AUXILIARY: *Harmony memory HM of size $m \times 2$, variables i, Acc, k, i_{min} and $minFitness$, and new harmony vector h of size 1×2.*

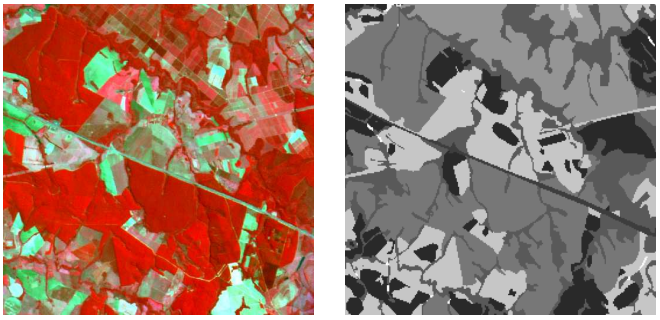
1. For each harmony i ($\forall i = 1, \ldots, hms$), do
2. $\quad HM_{i,1} \leftarrow$ Random(0,100).
3. $\quad [P_1, C_1, L_1, Z_1'] \leftarrow$Algorithm 4($Z_1$,$Z_2$,$HM_{i,1}$).
4. $\quad [L_2, P_2, Acc] \leftarrow$ Algorithm 2($[P_1, C_1, L_1, Z_1'], Z_2, (v, d)$).
5. $\quad HM_{i,j} \leftarrow Acc \left(\frac{1}{1+e^{-|Z_1'|}} \right)^{-1}$.
6. For $k \leftarrow 1$ to T, do
7. $\quad$ If ($hmcr > 0.5$), then
8. $\quad\quad i \leftarrow$ RandomInteger(1,hms).
9. $\quad\quad h[1] \leftarrow HM_{i,1}$.
10. $\quad\quad$ If ($par > 0.5$), then
11. $\quad\quad\quad h[1] \leftarrow h[1] + 0.1$.
12. $\quad$ Else $h[1] \leftarrow$ Random(0,100).
13. $\quad [P_1, C_1, L_1, Z_1'] \leftarrow$Algorithm 4($Z_1$,$Z_2$,$h[1]$).
14. $\quad [L_2, P_2, Acc] \leftarrow$ Algorithm 2($[P_1, C_1, L_1, Z_1'], Z_2, (v, d)$).
15. $\quad h[2] \leftarrow Acc \left(\frac{1}{1+e^{-|Z_1'|}} \right)^{-1}$.
16. $\quad [i_{min}, minFitness] \leftarrow$ GetMinFitness(HM).
17. $\quad$ If ($h[2] > minFitness$), then.
18. $\quad\quad HM_{imin,1} \leftarrow h[1]$.
19. $\quad\quad HM_{imin,2} \leftarrow h[2]$.
20. $i \leftarrow$ GetMaxHarmony(HM).
21. $MLoss \leftarrow HM_{i,1}$.
22. Returns $MLoss$.

V. EXPERIMENTS

In this section we described the experiments conducted, as well as the datasets employed in this work. For that, we used the LibOPF [10], which is a free library implemented in C language to the design of classifiers based on optimum-path forest. In regard to the datasets, we used data obtained from different applications, as described below:

- NTL: this dataset comprises with consumer profiles from a brazilian energy company, and was designed to identify thefts in power distribution systems, which are the so called non-technical losses (ntl). This dataset is composed by 736 samples, which are described by 4 features and distributed in 2 classes, i.e., illegal or legal consumer.
- CBERS: this dataset is composed by 25,876 samples, which represents pixels from an image obtained from CBERS-2B satellite. Each sample is represented by 21 features (18 texture features and 3 RGB values), distributed in 6 classes. Figure 1 displays the original and ground truth images.
- NSL-KDD[1]: this dataset is composed by 125,973 samples, which was designed for intrusion detection in computer networks. In that case, each sample is represented by 41 features extracted from connections to a local area network at Lincoln Labs - MIT (Massachusetts Institute of Technology). The number of classes is two, which stand for a normal access and an attack [11].
- IRIS: this dataset is one of the most used in the pattern recognition for benchmarking purposes, and it comprises 150 samples equally distributed in 3 classes. Each sample is represented by 4 features. The goal of this dataset it to distinguish three species of iris plant [12].
- MPEG-7: this dataset contains 1,400 images represented by their shapes, equally distributed in 70 classes [13]. In order to describe each sample, we used the Moments Invariants descriptor [14].

Fig. 1. Satellite images used in the experiments: covering the area of Itatinga, SP - Brazil by (a) CBERS-2B CCD sensor (2R3G4B) and its (b) respective ground truth image.



(a) (b)

[1]http://www.iscx.ca/NSL-KDD/

For each dataset, we used different percentages of training, evaluating and test sets, which were empirically chosen, based on our previous experience. Table I displays the percentages for each dataset. As aforementioned in Section IV, we used both training and evaluating sets to guide the whole process of finding optimal or near optimal values for $MLoss$ parameter, which aims to maximize the fitness function given by Equation 8.

TABLE I
PERCENTAGES USED FOR EACH DATASET.

Data	Training	Evaluating	Test
NTL	30%	20%	50%
CBERS	5%	45%	50%
NSL-KDD	1%	49%	50%
IRIS	30%	20%	50%
MPEG-7	20%	30%	50%

A. Results and Discussion

In this section, we described the experiments conducted over the five datasets: NTL, CBERS, NSL-KDD, IRIS and MPEG-7. In order to compare the proposed approach, we performed an exhaustive search for $MLoss$ within the interval $[0, 100]$. The values were sampled at 0.01, summarizing 100 values in the aforementioned interval. Figures 2, 3, 4, 5 and 6 display the curves for each dataset. The y axis stands for the fitness function F (Equation 8), while x axis denotes each $MLoss$ sampled value. In regard to HS parameters, we used $hms = 15$, $hmcr = 0.6$, $par = 0.3$ and $T = 5$ (number of iterations). Notice that these values were empirically chosen, based on our previous experience.

Fig. 2. Exhaustive search for $MLoss$ value over NTL dataset.

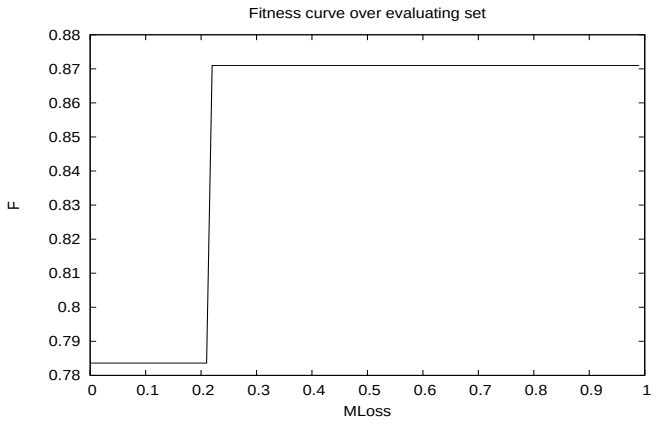


Fig. 3. Exhaustive search for $MLoss$ value over CBERS dataset.

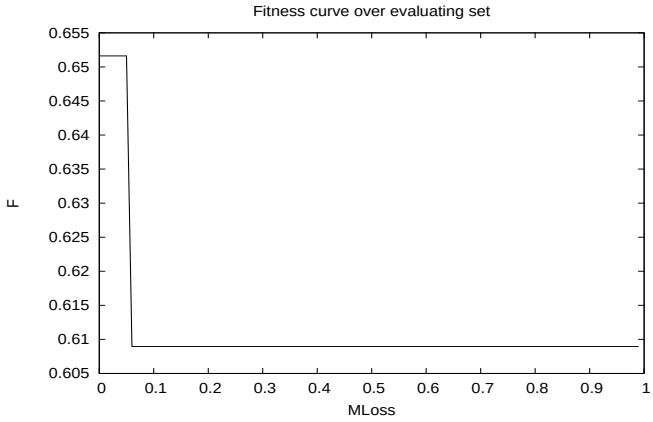


Fig. 4. Exhaustive search for $MLoss$ value over NSL-KDD dataset.

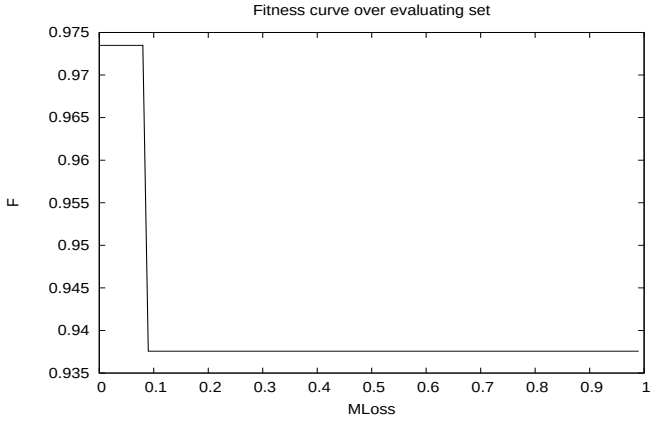


Fig. 5. Exhaustive search for $MLoss$ value over IRIS dataset.

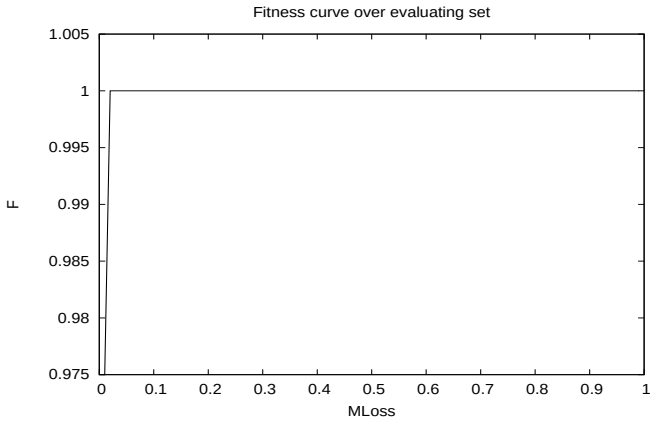


Table II displays the execution times in seconds regarding the exhaustive search for all datasets. We also showed the execution time of our proposed approach to find $MLoss$ automatically (seconds), which was described in Section IV. These results averaged over 10 runnings. One can also see the gain of the proposed technique in terms of execution time.

Table III displays the $MLoss$ values obtained through the proposed technique. One can see that for NTL and IRIS datasets these values correspond to the maximum of curves displayed in Figures 2 and 5, respectively. Although the values for MPEG-7, NSL-KDD and CBERS are not optimal, they are also suitable to accomplish the task, since a considerable amount of reduction in the training set has been accomplished, without affecting a lot the accuracy over the test set.

In addition, we would like to shed light over the choice

Fig. 6. Exhaustive search for $MLoss$ value over MPEG-7 dataset.

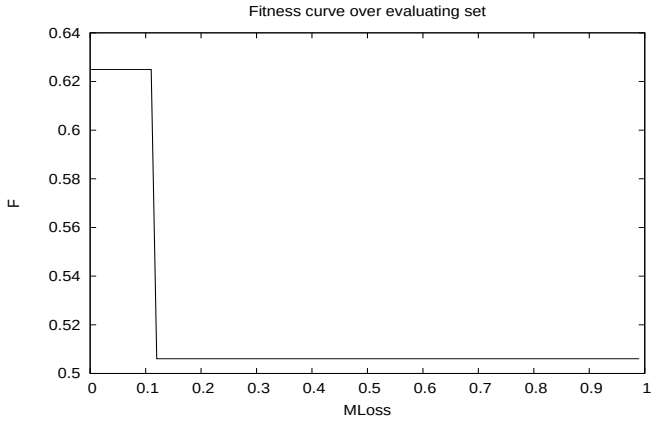


TABLE II
PERFORMANCES RESULTS: TIMINGS ARE EXPRESSED IN SECONDS.

Data	Exhaustive [s]	Proposed [s]	Gain
NTL	6.40 ± 0.49	1.27 ± 1.03	500%
CBERS	1984.10 ± 167.17	169.43 ± 507.01	1190%
NSL-KDD	21160.80 ± 2297.29	5260.90 ± 8035.36	920%
IRIS	1.70 ± 0.46	0.42 ± 0.06	370%
MPEG-7	8.20 ± 0.6	1.32 ± 1.05	6210%

for Harmony Search, which has been guided by its efficiency, since other widely used evolutionary-based techniques, such as Particle Swarm Optimization [15] and Gravitational Search Algorithm [16], have high computational burden. While the former technique attempts to move particles through the swarm by updating each dimension individually, the latter needs to sort the k best possible solutions to compute their new position, for each one of them.

VI. CONCLUSION

Optimum-path forest-based classifiers have been shown to be useful in several applications, that vary from remote sensing to biomedical signal recognition. The main skill for that classifiers is their fast training phases, even for large datasets. In regard to classification step, the computational complexity is proportional to the number of training samples, i.e., $\theta(|Z_1||Z_3|)$, in which Z_1 and Z_3 stand for training and test sets, respectively (similar definition is applied to evaluating set - Z_2). In order to make OPF faster, a training set pruning algorithm was developed, aiming to find the most relevant samples from training set, such that the remaining ones should be removed from that set. The idea is to design compact and representative training sets. This approach has been used in several applications, and its benefits can be highlighted in datasets in which one can find redundancy.

However, the main problem for OPF pruning algorithm is to set the $MLoss$ parameter, which is defined as the maximum allowed deviation of accuracy over the evaluating set with respect to the original and pruned training set. Thus, the user needs to define the allowed loss of accuracy. Henceforth, in some applications, the same user may not want to define that value. Thus, we proposed in this research a evolutionary-based algorithm to find propers values for $MLoss$. This algorithm is based on the Harmony Search, which is guided by a fitness function that considers the accuracy over the evaluating set and also the training set size, in order to find the best trade-off between effectiveness and efficiency for data classification.

TABLE III
VALUES FOR $MLoss$ FOUND BY THE PROPOSED APPROACH.

Data	$MLoss$
NTL	0.62
CBERS	0.62
NSL-KDD	0.91
IRIS	0.36
MPEG-7	0.19

We conducted experiments over 5 public datasets in order to show the robustness of our proposed hybrid approach (HS+OPF) against an exhaustive search for $MLoss$ values. We have seen that in 2 (NTL and IRIS) out of 5 datasets the proposed approach has found optimal $MLoss$ values, and with respect to the remaining 3 datasets, i..e, MPEG-7, NSL-KDD and CBERS, we have obtained good values. We also concluded that we can improve the results, but with the price of computational cost for that. For future works, we intent to make OPF pruning faster with GPGPU programming. Thus, the hybrid approach proposed here will be benefited with that.

ACKNOWLEDGMENT

The authors would like to thank FAPESP grants #2010/11676-7, #2010/02045-3 and #2009/16206-1.

REFERENCES

[1] J. P. Papa, A. X. Falcão, and C. T. N. Suzuki, "Supervised pattern classication based on optimum-path forest," *International Journal of Imaging Systems and Technology*, vol. 19, no. 2, pp. 120–131, 2009.

[2] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[3] J. P. Papa, A. X. Falcão, G. M. de Freitas, and A. M. H. de Ávila, "Robust pruning of training patterns for optimum-path forest classication applied to satellite-based rainfall occurrence estimation," *IEEE Geoscience and Remote Sensing Letters*, vol. 7, no. 2, pp. 396–400, 2010.

[4] R. J. Pisani, P. S. Riedel, A. R. Gomes, R. Y. M. Nakamura, and J. P. Papa, "Is is possible to make pixel-based radar image classification user-friendly?" in *Proceedings of the IEEE International Geoscience and Remote Sensing Symposium*, 2011, (accepted for publication).

[5] C. R. Pereira, R. Y. M. Nakamura, J. P. Papa, and K. A. P. Costa, "Intrusion detection system using optimum-path forest," in *Proceedings of the 36th Annual IEEE Conference on Local Computer Networks*, 2011, (submitted).

[6] Z. Geem, *Recent Advances In Harmony Search Algorithm*, ser. Studies in Computational Intelligence. Springer, 2010, vol. 270.

[7] E. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[8] A. X. Falcão, J. Stolfi, and R. A. Lotufo, "The image foresting transform theory, algorithms, and applications," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 1, pp. 19–29, 2004.

[9] C. Allène, J. Y. Audibert, M. Couprie, J. Cousty, and R. Keriven, "Some links between min-cuts, optimal spanning forests and watersheds," in *Mathematical Morphology and its Applications to Image and Signal Processing (ISMM'07)*. MCT/INPE, 2007, pp. 253–264.

[10] J. P. Papa, C. T. N. Suzuki, and A. X. Falcão, *LibOPF: A library for the design of optimum-path forest classifiers*, 2009, software version 2.0 available at http://www.ic.unicamp.br/~afalcao/LibOPF.

[11] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, “A detailed analysis of the kdd cup 99 data set,” in *Proceedings of the Second IEEE international conference on Computational intelligence for security and defense applications*. Piscataway, NJ, USA: IEEE Press, 2009, pp. 53–58.

[12] A. Frank and A. Asuncion, “UCI machine learning repository,” 2010. [Online]. Available: http://archive.ics.uci.edu/ml

[13] MPEG-7, “Mpeg-7: The generic multimedia content description standard, part 1,” *IEEE MultiMedia*, vol. 09, no. 2, pp. 78–87, 2002.

[14] M. Hu, “Visual Pattern Recognition by Moment Invariants,” *IRE Transactions on Information Theory*, vol. 8, no. 2, pp. 179–187, 1962.

[15] J. Kennedy and R. C. Eberhart, *Swarm intelligence*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001.

[16] E. Rashedi, H. Nezamabadi-pour, and S. Saryazdi, “Gsa: A gravitational search algorithm,” *Information Sciences*, vol. 179, no. 13, pp. 2232–2248, June 2009.