2014 International Joint Conference on Neural Networks (IJCNN)
July 6-11, 2014, Beijing, China

A flocking-like technique to perform semi-supervised learning

Roberto A. Gueleri*, Thiago H. Cupertino*, André C.P.L.F. de Carvalho* and Liang Zhao†
* Institute of Mathematics and Computer Science – University of São Paulo – São Carlos – SP – Brazil
r.a.gueleri@gmail.com, gueleri@icmc.usp.br (Roberto)
thcupertino@gmail.com, thiagohc@icmc.usp.br (Thiago), andre@icmc.usp.br (André)
† Ribeirão Preto School of Philosophy, Science and Literature – University of São Paulo – Ribeirão Preto – SP – Brazil
zhao@usp.br (Liang)

***Abstract*—We present a nature-inspired semi-supervised learning technique based on the flocking formation of certain living species like birds and fishes. Each data item is treated as an individual in the flock. Starting from random directions, each data item moves according to its surrounding items, by getting closer to them (but not too much close) and taking the same direction of motion. Labeled items play special roles, ensuring that data from different classes will belong to different, distant flocks. Experiments on both artificial and benchmark datasets were performed and show its classification accuracy. Despite the rich behavior, we argue that this technique has a sub-quadratic asymptotic time complexity, thus being feasible to be used on large datasets. In order to achieve such performance, a space-partitioning technique is introduced. We also argue that the richness behind this dynamic, self-organizing model is quite robust and may be used to do much more than simply propagating the labels from labeled to unlabeled data. It could be used to determine class overlapping, wrong labeling, etc.**

I. Introduction

Semi-supervised learning is a machine learning task designated to work on datasets having most of their items unclassified (unlabeled) and a few of them previously classified (labeled) [1], [2], [3], [4]. Specifically, *transductive learning* aims to classify the unlabeled data according to the labeled ones. *Inductive learning*, in turn, aims to predict a classification function instead of just classifying the existent unlabeled data. Besides the previously classified data, in both tasks (transductive and inductive) the pattern of the large amount of unclassified data can also provide valuable information, i.e., the unlabeled data can be used to classify themselves (transductive task) or to output the classification function (inductive task).

In this paper, we present a nature-inspired transductive learning technique, based on the flocking formation of certain living species like birds and fishes. Those systems are able to display a complex global behavior without any leadership [5], [6]. The coordinated, collective motion just *emerges* from the local interactions among the individuals that make up the system. Such self-organization is a powerful feature that can be explored on machine learning problems, because of its intrinsic ability on dealing with imprecision. Precisely, on semi-supervised learning problems, we want systems that are resilient to noise, wrong labeling, class overlapping, etc.

Cui et al. [7] have already proposed a flocking-based technique for document clustering analysis. Cui's work and ours are both based on the Reynolds' *boids* model [8], [9]. The flocking formation is achieved by modeling three main interaction rules: *alignment*, *approximation* (cohesion), and *separation*. Despite these similarities however, the simple fact that Cui's technique aims at unsupervised learning (clustering) leads by itself to many modeling differences in comparison to our semi-supervised learning technique. In our model, labeled items play special roles by pushing away items that belong to different classes. As a result, it is expected that items belonging to different classes will make up different, distant flocks.

We also introduce a space-partitioning technique used to improve performance. Doing so, we argue that our flocking-like semi-supervised learning technique has a sub-quadratic time complexity. Some preliminary results were obtained by running our technique on artificial and benchmark datasets. The classification accuracy obtained is comparable to those from other well-known techniques. So we achieve a balance between good classification and acceptable computational cost. More importantly, we try to illustrate that the richness behind this flocking-like approach can be used to do more complex tasks such as soft labeling.

The rest of the paper is organized as follows. Sec. II formally describes our proposed model. Since computational cost is a concern, Sec. III discusses the computational complexity of our proposed model and presents a space-partitioning technique used to improve performance. Sec. IV presents experimental results performed on artificial and benchmark datasets. Running our technique on benchmark datasets has enabled a comparison to other well-known techniques from the literature. Finally, Sec. V concludes the paper.

II. Model description

In this section, we formally describe our flocking-like semi-supervised learning technique. Let us begin, however, with just an illustration of how the proposed dynamical model behaves. Fig. 1 displays the formation of flocks on an artificial dataset. The overall idea is that at the beginning, every data item is assigned a random velocity. Actually, the speed of every item remains constant throughout the process. Only the direction is changed. This is a common approach on modeling flocking systems, as the well-known Vicsek's work [5]. As the dynamical process evolves, the new velocity of each item is a function of the location and velocity of its surrounding items. They all tend to get closer to each other, but not too much close. Also, they tend to get the same direction of motion.

978-1-4799-1484-5/14/$31.00 ©2014 IEEE

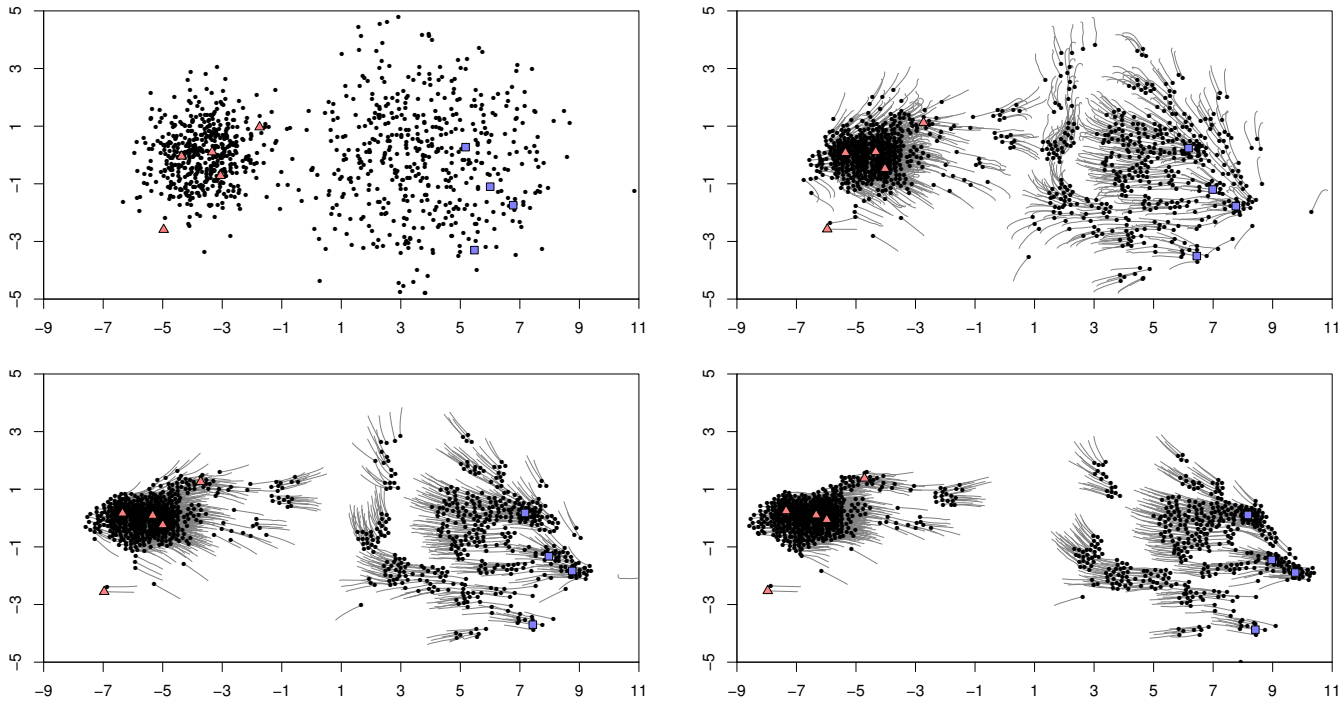

Fig. 1. Illustration showing the formation of flocks. [Top-left:] At $t = 0$, the original dataset. A few initially labeled items are denoted by special marks: triangles and squares. [Top-right, bottom-left, and bottom-right:] Respectively, at $t = 100$, $t = 200$, and $t = 300$, each one displaying the trajectories of all the items along the last 100 iterations. Horizontal and vertical axes represent the values of first and second data attributes, respectively.

The closer two items are, the stronger the influence over each other. Labeled items play special roles, specially regarding the interaction between two items having distinct labels: they tend to get away from each other. This feature raises a competition for label propagation.

Now let us turn to a formal description. The dataset is composed of two sub-sets: $\mathcal{U}$ of unlabeled and $\mathcal{L}$ of labeled items. Each data item $\mathbf{x}_i$ is a vector in the feature space. N is the dataset size (number of unlabeled plus labeled items) and D is the dimensionality of the feature space, i.e., the number of features that characterizes each item.

Let $\mathbf{x}_i(t)$ be the position of the i-th data item at iteration (time) t, and let $\mathbf{v}_i(t)$ be the corresponding velocity of that item. Thus the data motion is given by the following equations:

$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \alpha_1 \cdot \frac{\mathbf{v}_i(t)}{\|\mathbf{v}_i(t)\|} \tag{1}$$

$$\mathbf{v}_i(t+1) = \frac{\mathbf{v}_i(t)}{\|\mathbf{v}_i(t)\|} + \alpha_2 \cdot \frac{\mathbf{v}_i^{\text{chg}}(t)}{\|\mathbf{v}_i^{\text{chg}}(t)\|} \tag{2}$$

$$\begin{aligned}\mathbf{v}_i^{\text{chg}}(t) = \sum_{\substack{\mathbf{x}_j \mid \mathbf{x}_j \in \text{nei}(\mathbf{x}_i,t) \wedge \\ \neg[\mathbf{x}_i,\mathbf{x}_j \in \mathcal{L} \wedge \\ \text{class}(\mathbf{x}_i) \neq \text{class}(\mathbf{x}_j)]}} \text{type}(\mathbf{x}_i,\mathbf{x}_j) \cdot \text{balance}(\mathbf{x}_i,\mathbf{x}_j) \cdot \Bigg[\\ \text{align} \cdot \frac{\mathbf{v}_j(t)}{\|\mathbf{v}_j(t)\|} \cdot \frac{1}{\|\mathbf{x}_i - \mathbf{x}_j\|^{\beta_1}} + \\ + \text{approx} \cdot \frac{\mathbf{x}_j - \mathbf{x}_i}{\|\mathbf{x}_j - \mathbf{x}_i\|} \cdot \frac{1}{\|\mathbf{x}_i - \mathbf{x}_j\|^{\beta_2}} + \\ + \text{sep} \cdot \frac{\mathbf{x}_i - \mathbf{x}_j}{\|\mathbf{x}_i - \mathbf{x}_j\|} \cdot \frac{1}{\|\mathbf{x}_i - \mathbf{x}_j\|^{\beta_3}} \Bigg] + \\ + \sum_{\substack{\mathbf{x}_j \mid \mathbf{x}_i,\mathbf{x}_j \in \mathcal{L} \wedge \\ \text{class}(\mathbf{x}_i) \neq \text{class}(\mathbf{x}_j)}} \text{type}(\mathbf{x}_i,\mathbf{x}_j) \cdot \frac{\mathbf{x}_i - \mathbf{x}_j}{\|\mathbf{x}_i - \mathbf{x}_j\|}\end{aligned} \tag{3}$$

One may notice that the velocity is always used as an unit vector ($\mathbf{v}_i(t)/\|\mathbf{v}_i(t)\|$). In Eq. 1, it reflects the constant-speed motion. Eq. 3 is just an auxiliary equation giving $\mathbf{v}_i^{\text{chg}}(t)$, the *change* of velocity of the i-th data item at iteration t. Its value is also normalized when used in Eq. 2 ($\mathbf{v}_i^{\text{chg}}(t)/\|\mathbf{v}_i^{\text{chg}}(t)\|$). This latter normalization makes its usage independent of the dataset size, since this size affects directly the resultant value $\mathbf{v}_i^{\text{chg}}$.

The second summation in Eq. 3 takes place over all pairs $(\mathbf{x}_i, \mathbf{x}_j)$ such that both are labeled and belong to different classes. It drives items from different classes to take distinct directions. The first summation, in turn, enables the expected flocking behavior and acts over all the other pairs $(\mathbf{x}_i, \mathbf{x}_j)$, except those which both $\mathbf{x}_i$ and $\mathbf{x}_j$ are unlabeled and $\mathbf{x}_j$ does not belong to the neighborhood of $\mathbf{x}_i$. This filtering is denoted

TABLE I. VALUES ASSIGNED FROM $\mathrm{type}(\mathbf{x}_i, \mathbf{x}_j)$.

$\mathbf{x}_i$	$\mathbf{x}_j$	**Value used in the experiments**
unlabeled	unlabeled	1
unlabeled	labeled	20, 50 (*)
labeled	unlabeled	0.05
labeled	labeled (same class)	10
labeled	labeled (different classes)	10

(*) Values used on artificial and benchmark datasets, respectively.

by the function $\mathrm{nei}(\mathbf{x}_i, t)$, which means the *neighborhood* of $\mathbf{x}_i$ at iteration t. It must be highlighted that such filtering applies only when both $\mathbf{x}_i$ and $\mathbf{x}_j$ are unlabeled. Labeled items have "greater importance" and have always to be computed, no matter how distant they are. Since unlabeled items are found in much more number than labeled ones (and so the number of unlabeled-unlabeled pairs), and since the nearest neighbors are those items that dominate the resultant value $\mathbf{v}_i^{\mathrm{chg}}$, doing such filtering reduces drastically the computational cost without deviating too much from the ideal result when considering the interaction between all the pairs of items. All the experiments presented here consider only the five nearest neighbors. In Sec. III, we present a space-partitioning technique that avoids the N^2 cost of determining the nearest neighbors at each time step.

Inside the first summation in Eq. 3, there are three main terms each being led by the coefficients align, approx, and sep. As these names suggest, such terms refer to the composition of a flocking behavior by combining three components: alignment, approximation, and separation. The coefficients align, approx, and sep are just parameters of the model. The term $\mathbf{x}_j - \mathbf{x}_i / \|\mathbf{x}_j - \mathbf{x}_i\|$ is a unit vector that gives the direction from $\mathbf{x}_i$ to $\mathbf{x}_j$. Compare it to $\mathbf{x}_i - \mathbf{x}_j / \|\mathbf{x}_i - \mathbf{x}_j\|$, which gives the opposite direction. The term $1/\|\mathbf{x}_i - \mathbf{x}_j\|^*$, where $*$ assumes β_1, β_2, or β_3, results in a factor that depends on the distance between $\mathbf{x}_i$ and $\mathbf{x}_j$. The closer they are, the more $\mathbf{x}_j$ is influencing the motion of $\mathbf{x}_i$. β_1, β_2, and β_3 are parameters. All of them must be positive, and β_3 must be greater than β_2 in order to achieve the proper equilibrium between approximation and separation. In the experiments presented here, we set $\beta_1 = \beta_2$ and $\beta_3 = \beta_2 + 1$.

The dynamical model proposed here is made up by computing the interaction between pairs of data items. Each pair can be distinguished by the type of its items. Doing so enables us to, for instance, set a labeled item to have more influence than an unlabeled one. Therefore, each pair $(\mathbf{x}_i, \mathbf{x}_j)$ is assigned a factor that quantifies the influence of $\mathbf{x}_j$ over the motion of $\mathbf{x}_i$. Such factor is denoted by the function $\mathrm{type}(\mathbf{x}_i, \mathbf{x}_j)$ (Eq. 3), which assigns values according to Table I. Of course, we are free to modify those values in order to get different behaviors. Those values set in Table I are just the values used in our experiments.

Datasets may be unbalanced, not only in terms of their size but also in terms of the amount of labeled items. Regarding the number of labeled items, a class that has the majority of them tend to absorb most of the unlabeled ones. It happens just because each labeled item applies a force on every unlabeled, thus making such biggest class also the strongest one. To avoid this situation, we need an artifact to balance such forces. The function $\mathrm{balance}(\mathbf{x}_i, \mathbf{x}_j)$, in Eq. 3, does this job. Along the two-class experiments presented in this paper, $\mathrm{balance}(\mathbf{x}_i, \mathbf{x}_j)$ was set as:

$$\mathrm{balance}(\mathbf{x}_i, \mathbf{x}_j) = \begin{cases} |\mathcal{L}_1|/|\mathcal{L}_2| & \text{if } \mathbf{x}_j \in \mathcal{L}_2 \wedge \mathbf{x}_i \in \mathcal{U} \\ |\mathcal{L}_2|/|\mathcal{L}_1| & \text{if } \mathbf{x}_j \in \mathcal{L}_1 \wedge \mathbf{x}_i \in \mathcal{U} \\ 1 & \text{if } \mathbf{x}_i \in \mathcal{L} \end{cases} \quad (4)$$

where $\mathcal{L}_1$ and $\mathcal{L}_2$ are the sets of labeled items that belong to classes 1 and 2, respectively, and $|\mathcal{L}_1|$ and $|\mathcal{L}_2|$ are their sizes. When $\mathbf{x}_i$ is labeled, $\mathrm{balance}(\mathbf{x}_i, \mathbf{x}_j)$ has no effect. Generalizations can also be made to support datasets with more than two classes.

Finally, there are two more parameters to be explained: α_1 and α_2 (Eqs. 1 and 2). They can be seen as the precision of the dynamical process: the less their values, the smoother the dynamics, but the longer the time to reach a stable configuration (regarding the flocks that emerge). However, some care must be taken when setting them, because they are not independent from each other. They actually set the rate by which the position and velocity of the data items change. For instance, if we set the effect of α_1 much more intense than α_2, then the items may travel large distances with virtually no change on the direction of motion.

III. COMPUTATIONAL COMPLEXITY

In this section, we discuss the computational cost of the proposed technique. Specifically, we are going to talk about the time complexity. The data structures presented in this section do not introduce significant space (memory) consumption, so the space complexity will remain on the order of N, the dataset size.

At each time step, a process like this one would have a cost of N^2, save for the filtering by considering only the interaction among the nearest neighbors, instead of an all-with-all interaction. However, the naive procedure of finding such neighbors would by itself have a cost of N^2. So we introduce a space-partitioning technique that is expected to avoid such a high cost. For simplification purposes, we are not considering that labeled and unlabeled items play different roles. Since the labeled data is much fewer than the unlabeled, such simplification has no major implications on the following analysis.

The partition of the feature space results in a hierarchy of regions, denoted by a tree. The root node represents the largest region being considered, large enough to contain all the dataset, and its children represent sub-regions that contain only a sub-set of the data. The tree ends in leaf nodes each containing just one data item. There is a restriction that no two data items are allowed to occupy the exact same position (actually a duplicate item), otherwise the process will fail, just because no partition could take them apart. Fig. 2 displays how the space partitioning looks like.

As the dynamical process goes on, at each time step all the tree has to be rebuilt to keep track of the new data configuration. The neighbors of an item may change (and probably will) as the system evolves. We can break the procedure of finding the neighbors into two sub-procedures: (1) building the tree and (2) walking the tree to find the neighbors of each item.

The first process is described as follows:

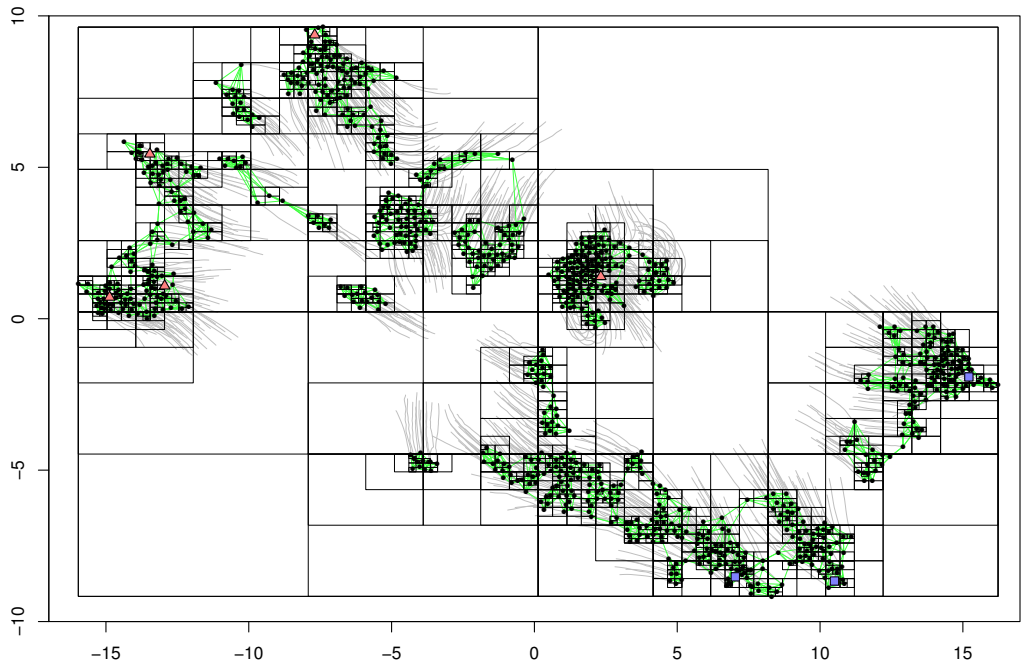

Fig. 2. How the space partitioning looks like in a two-dimensional space. Rectangles represent the nodes (orthants) of the tree. The biggest rectangle is large enough to cover all the items and, recursively, it is broken into smaller ones whenever there are more than one item inside it. Notice that every item lies inside its own rectangle, which can be seen as its address in the space-partitioning tree. Labeled items are much fewer and play special roles, so they are not accounted for the partitioning. Links between each item and its five closest neighbors are also displayed (in green), as well as the trajectories along the last iterations (in gray). Horizontal and vertical axes represent the values of first and second data attributes, respectively.

1) Start with a region in the feature space such that each dimension is bounded and big enough just to cover the entire dataset. This is the root node and it links to all the data items.
2) Divide that region into 2^D new sub-regions by halving each dimension. Let us call each new sub-region as an *orthant* (generalization of *quadrants* and *octants*).
3) Recursively divide (Step 2) each orthant whenever it covers more than one data item.

Each node (orthant) must link to all the data it covers. It also has to keep track of its region boundaries.

A rather balanced tree will have a height around $\log_{(2^D)} N$ (base is 2^D), which is smaller than $\log_2 N$. So the cost of building that tree has an order of $N \cdot \log_{(2^D)} N$.

Now we describe how to find the K nearest neighbors of an item $\mathbf{x}_i$, by walking the previously built tree:

1) Starting from the root node, go down the tree until the leaf node that covers $\mathbf{x}_i$ is reached. Doing so is straightforward, we just have to look at the boundaries of each orthant and check whether $\mathbf{x}_i$ is inside, then repeating this process on the sub-orthants and so on. We could also make a link from $\mathbf{x}_i$ to its leaf node, thus avoiding this step but consuming more memory. On a somewhat balanced tree, however, this searching process has a cost of order $\log_{(2^D)} N$, thus pretty fast.
2) Once the leaf node is found, go one level up (parent node) and measure the distance between $\mathbf{x}_i$ and all the other items $\mathbf{x}_j$ covered by this parent node, selecting the K nearest ones. Remember that each node links to all the data it covers, so only a very small set of items has to be visited. If there are fewer data than the K desired number of neighbors, then go up the tree one level further.
3) Once the K $\mathbf{x}_j$ neighbors were selected, we have to check whether there could be another unvisited item, say $\mathbf{x}_k$, such that $\mathbf{x}_k$ is closer to $\mathbf{x}_i$ than some of $\mathbf{x}_j$. To do so, we check if the distance between $\mathbf{x}_i$ and the farthest $\mathbf{x}_j$ ("last neighbor") is greater than the distance between $\mathbf{x}_i$ and the boundaries of the visited region (the highest visited orthant; remember the nodes carry such information). If so, then we have to visit one node above and repeat the process. Otherwise, the neighbors we found are guaranteed to be the closest ones. Determining precisely the computational cost of this step is a hard task. Furthermore, it depends on D and K. On average, however, it seems like the number of visited items is much less than the total number N. So we can say that the cost of finding the nearest neighbors of a given item has an order less than N (even considering Step 1), consequently the cost of finding the neighbors of all the data is less than N^2.

Placing the two procedures together (building and walking the tree), the total time cost remains below N^2. This is the total cost of finding the K nearest neighbors of every item, assuming K is sufficiently small.

Now we have to consider the cost of computing the

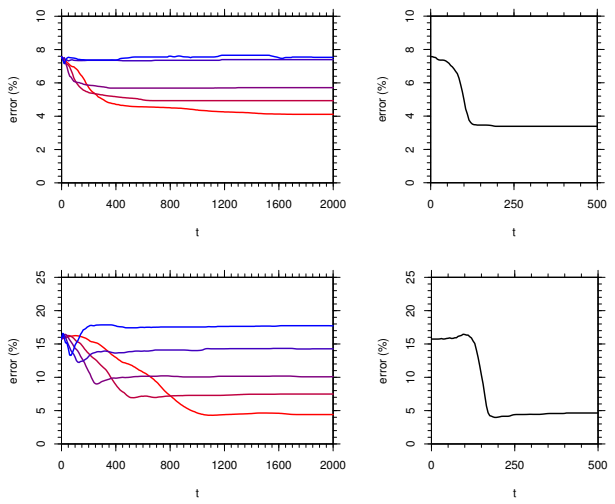

Fig. 3. Dependency of classification accuracy on α_1 and on number of iterations t. [Top:] Two-gaussians dataset (see Fig. 4). [Bottom:] Interlaced-arcs dataset (Fig. 4). [Left:] Constant value of α_1 throughout the process. Experiments performed on five different values of α_1. From red to blue: 0.005, 0.01, 0.02, 0.04, and 0.08 on two-gaussians dataset [top-left], and 0.01, 0.02, 0.04, 0.08, and 0.16 on interlaced-arcs dataset [bottom-left]. Each graph is the average of 30 simulations, and measured at every 10 time steps. [Right:] Experiments using incremental values of α_1. On both datasets [top-right and bottom-right], it starts on 0.001 and is multiplied by a factor of 1.2 at every 5 steps, until it reaches a maximum value of 0.2. Again, each graph is the average of 30 simulations, but now measured at every 5 time steps.

interactions among the data items, as described in Sec. II, Eqs. 1, 2, and 3. Since we are interested in the asymptotic cost, we just have to look at the summations of Eq. 3. At each time step, each item $\mathbf{x}_i$ is interacting with its K neighbors $\mathbf{x}_j$. Since K is a small constant, computing the interactions among all the data has just a cost of order N, thus linear. This plus the cost of finding the neighbors still results in a sub-N^2 cost.

Finally, we have to bear in mind that the previous analysis holds only for a single time step. The final question is: how many steps would be necessary to achieve good classification results? This is a much more intriguing question and a comprehensive analysis is beyond the scope of this paper. However, some empirical results were obtained.

Fig. 3 displays the dependency of classification accuracy on α_1 and on number of iterations t. Parameter α_2 was set to 1 on all simulations. Other parameters were set as discussed in the next section (Sec. IV). At any given time, the classification is obtained by assigning every unlabeled item the label of the closest labeled item. Notice that at $t = 0$, the result is just the same as using 1-Nearest Neighbor (1-NN) classification technique. We clearly see that smaller values of α_1 (finer precision) lead to better classification accuracies (left-hand figures). We may hypothesize that the cause of poor results using larger values of α_1 resides just on the beginning of the process. That is because the data items start with random directions, and large steps using random directions may degrade the dataset pattern, mixing up data belonging to different classes. So we also evaluate a variant of the model, using incremental values of α_1, so that it starts very small and, as the system achieves some stability (at least regarding to the directions of the items), it gets larger. These results is displayed on the right-hand side of Fig. 3. On both datsets, we got results as good as using small values of α_1, but now demanding much less time steps.

We also expect that the number of steps needed to achieve good classification results does not scale with the number of items N, at least not linearly. The dependency of classification quality on t seems to lie on the geometry of the dataset, not directly on N. Assuming it is true, we can still stating that the proposed technique has a sub-N^2 time cost, now considering the entire process. However, a further analysis on how the number of steps scales with N has yet to be done.

IV. Experimental results

In order to evaluate the classification accuracy of the proposed model, we now introduce experimental results performed over both artificial and benchmark datasets. Firstly, let us present the overall parameter setting. The factor $\mathrm{type}(\mathbf{x}_i, \mathbf{x}_j)$ (Eq. 3, Sec. II) was set according to Table I (Sec. II). Setting parameter α_1 depends on the dataset density. We used values ranging from 0.01 for more compact datasets, to 0.04 for sparser ones. α_2 was just set to 1. β_1 was set to 2 on artificial datasets and from 2 to 10 on benchmark datasets. β_2 and β_3 were set as $\beta_2 = \beta_1$ and $\beta_3 = \beta_2 + 1$. Only the five nearest neighbors was used to compute the interaction between unlabeled items.

By looking at Eq. 3, one can see that if we isolate the approximation and separation terms, there will be a distance $\|\mathbf{x}_i - \mathbf{x}_j\|$ by which the approximation and separation terms neutralize each other. That is the idea behind the flocking formation: "two data items tend to get close to each other, but not too much close". Thus, instead of setting approx and sep constants independently, we may want to set the desired equilibrium distance d_{eq}. Assuming $\beta_3 = \beta_2 + 1$, then we just set $\mathrm{approx} = 1$ and it results that $\mathrm{sep} = d_{\mathrm{eq}}$. In order to reduce the dependency of parameter adjustment on the dataset, in all the simulations we set d_{eq} to be the average of the distances between every unlabeled item and its closest neighbor. Finally, the parameter align was just set to 1.

A. Artificial datasets

Fig. 4 presents a classification map on two artificial datasets. Each dataset has 1000 data items, a few of them initially labeled and denoted by special marks: triangles and squares. Each color denotes one class: red for "triangle" and blue for "square". Each entry on this map was made by placing one more unlabeled item at that location, then running the process and checking which label that item gets. The label it gets is just the label of the closest labeled item, like an 1-NN classification, but measured after some time evolution (750 steps for the two-gaussians and 1000 steps for the interlaced-arcs datasets). Due to assignment of random initial direction for all the data items, each map entry is averaged 15 times by running 15 entire simulations. Values in-between red and blue mean that the additional unlabeled item was classified sometimes as red-class and sometimes as blue-class. $\alpha_1 = 0.01$ for the two-gaussians and $\alpha_1 = 0.02$ for the interlaced-arcs datasets. As a comparison, Fig. 4 gives also the classification map by using the 1-NN technique.

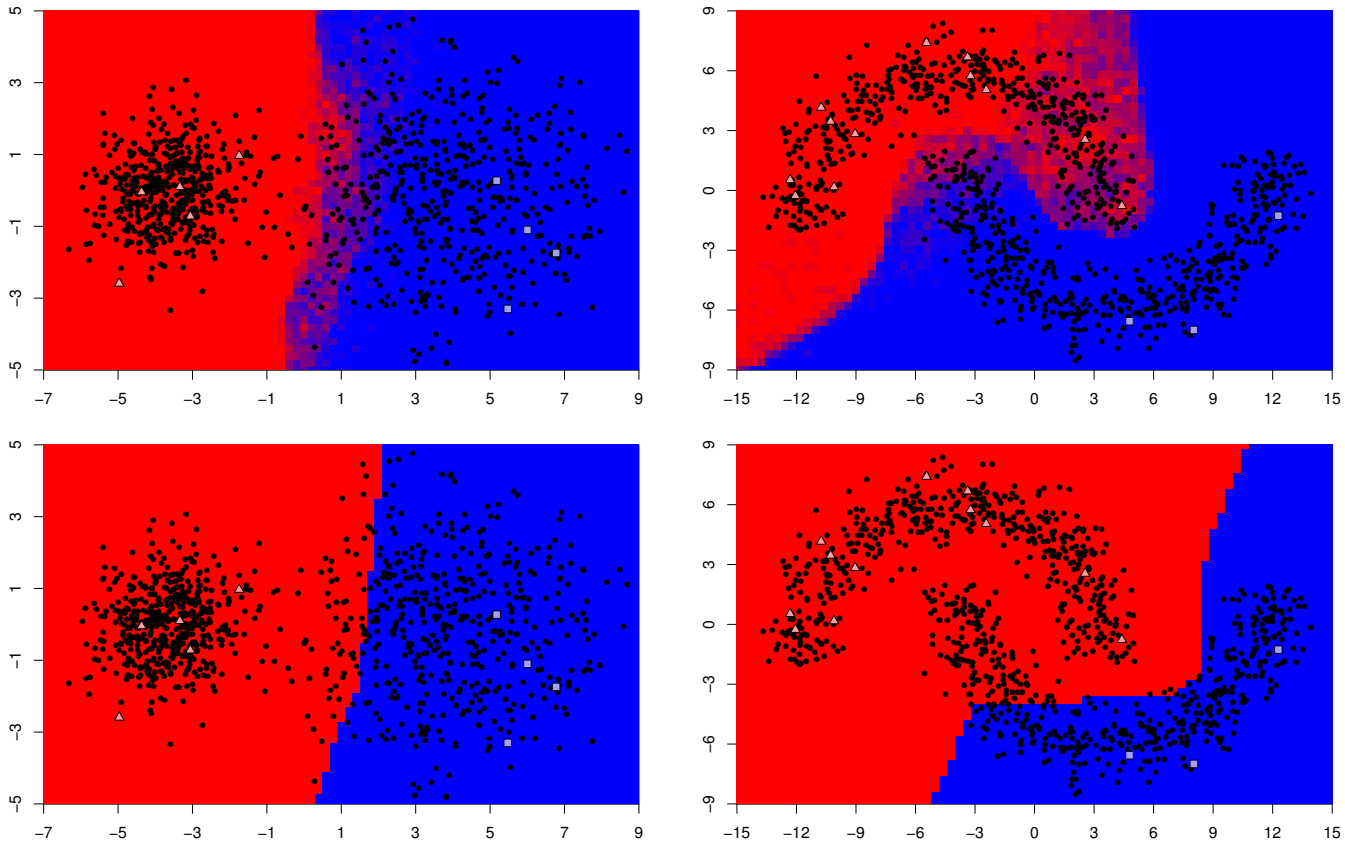

Fig. 4. Classification map on two artificial datasets: “two gaussians” [left] and “interlaced arcs” [right]. [Top:] Using the proposed technique. Each entry is the average of 15 simulations, each one measured after 750 time steps on “two gaussians”, and 1000 time steps on “interlaced arcs”. [Bottom:] Using 1-NN. Horizontal and vertical axes represent the values of first and second data attributes, respectively.

Although 1-NN is much faster, the results from the proposed technique are much more consistent. Moreover, looking at the classification maps allows determining regions of difficult labeling (colors in-between red and blue). This feature could be used for *soft labeling* other than just assigning a label, i.e., items could be assigned a level of membership in each class. In practice, for a transductive learning task, such maps are not even needed to perform a soft classification. We just have to run the technique a number of times, using different random direction initializations, and then compare how many times a given unlabeled item gets classified as one class or another.

B. Benchmark datasets

In order to compare the performance of the proposed technique to other well-known techniques from the literature, we have conducted experiments on benchmark datasets. Table II displays the datasets we employed, as proposed by Chapelle et al. [1]. Each dataset is available in 12 different versions, called *splits*. The data items are exactly the same, but each split has a different set of initially labeled items.

Table III displays the other techniques from the literature, used for classification accuracy comparison. The respective reference for each technique is also given.

Table IV presents the classification accuracy comparison. The datasets were pre-processed by using the first five PCA components, except for MVU and LEM, which are dimensionality-reduction techniques. The results of the other techniques from the literature were obtained from Cupertino

TABLE II. THE BENCHMARK DATASETS USED FOR THE EXPERIMENTS, PROPOSED BY CHAPELLE ET AL. [1].

Dataset	Classes	Dimensions (*)	Size
g241c	2	241	1500
g241d	2	241	1500
Digit1	2	241	1500
USPS	2	241	1500
BCI	2	117	400

(*) All the results presented in this paper, including the other techniques from the literature, was achieved by using only the first five PCA components.

TABLE III. REFERENCES FOR THE OTHER TECHNIQUES USED FOR COMPARISON.

Abbreviation	Technique	Ref.
k-NN	k-Nearest Neighbors	[10]
MVU	Maximum Variance Unfolding	[11]
LEM	Laplacian Eigenmaps	[12], [13], [14], [15], [16]
SVM	Support Vector Machines	[17]
TSVM	Transductive Support Vector Machines	[18], [19], [2]
SGT	Spectral Graph Transducer	[20]
LDS	Low-Density Separation	[18]
Lapl. RLS	Laplacian Regularized Least Squares	[21], [13], [15], [16]
Attr. Forces	Attraction Forces	[4]

TABLE IV. CLASSIFICATION ERRORS (%) ON THE DATASETS PROPOSED BY CHAPELLE ET AL. [1]. THE DATASETS WERE PRE-PROCESSED BY USING THE FIRST FIVE PCA COMPONENTS (EXCEPT FOR MVU AND LEM, WHICH ARE DIMENSIONALITY-REDUCTION TECHNIQUES). THE RESULTS OF THE OTHER TECHNIQUES FROM THE LITERATURE WAS OBTAINED FROM CUPERTINO ET AL. [4]. DIFFERENT COMBINATIONS OF β_1, β_2, AND β_3 WERE EVALUATED. ON ALL OF THEM, β_2 AND β_3 WERE SET AS $\beta_2 = \beta_1$, AND $\beta_3 = \beta_2 + 1$. THE BEST RESULTS FROM THE LITERATURE ARE HIGHLIGHTED, AS WELL AS THE BEST RESULTS OF OUR PROPOSED TECHNIQUE. THE RANK OF OUR TECHNIQUE (AMONG 13) IS ALSO DISPLAYED. ENTRIES MARKED WITH "-" WERE NOT SIMULATED.

	10 initially labeled items					100 initially labeled items					
	g241c	**g241d**	**Digit1**	**USPS**	**BCI**	**g241c**	**g241d**	**Digit1**	**USPS**	**BCI**	
1-NN	28.95	27.20	21.77	23.93	49.10	21.40	10.98	7.04	13.60	47.36	
MVU + 1-NN	47.15	45.56	14.42	23.34	47.95	43.01	38.20	2.83	6.50	47.89	
LEM + 1-NN	44.05	43.22	23.47	19.82	48.74	40.28	37.49	6.12	7.64	44.83	
SVM (linear)	24.00	28.67	21.53	31.52	48.50	13.34	13.99	6.79	21.60	46.92	
SVM (RBF)	44.41	39.23	33.15	19.69	49.79	17.76	9.12	5.73	13.68	48.06	
SVM (quad)	39.33	31.95	26.62	25.62	49.64	17.56	10.41	6.18	12.88	49.00	
SVM (poly)	33.47	33.23	29.63	26.09	49.44	22.05	13.48	7.40	18.45	47.75	
TSVM	21.38	31.23	20.32	30.04	48.96	13.77	21.63	9.15	14.54	45.65	
SGT	17.83	12.94	14.63	26.50	49.64	14.11	6.72	4.91	13.57	48.50	
LDS	17.46	36.48	18.90	23.48	49.02	13.19	8.39	4.49	19.33	47.31	
Laplacian RLS	27.10	26.67	18.03	18.68	49.38	16.04	7.83	4.88	11.76	46.11	
Attr. Forces	27.64	27.42	18.31	18.31	49.64	21.63	10.24	6.41	12.44	45.63	
	35.08	34.70	23.84	70.21	49.90	20.02	12.47	8.11	78.99	48.64	$\beta_1 = 2$
	29.48	29.80	22.57	55.15	49.81	17.24	9.43	6.44	58.04	47.99	$\beta_1 = 3$
	27.57	27.98	21.88	46.44	49.87	17.17	8.80	6.20	40.70	47.69	$\beta_1 = 4$
Our technique	27.00	27.72	23.37	39.29	49.71	17.51	8.75	6.41	30.27	47.50	$\beta_1 = 5$
	27.24	28.16	26.28	35.72	49.75	17.90	8.97	6.98	25.14	47.38	$\beta_1 = 6$
	-	-	-	35.52	49.72	-	-	-	20.43	47.54	$\beta_1 = 8$
	-	-	-	36.75	49.86	-	-	-	19.48	47.03	$\beta_1 = 10$
Our rank (/13)	5	5	9	13	12	6	4	8	12	6	

et al. [4]. Results on the original datasets, without PCA pre-processing, can be found in Chapelle's book [1]. We set $\alpha_1 = 0.01$ for *Digit1*, $\alpha_1 = 0.02$ for *g241c*, *g241d*, and *USPS*, and $\alpha_1 = 0.04$ for *BCI* datasets. Measures were made after 750 time steps. Each entry on the table (for our technique) is the average of 12 splits $\times$ 10 executions per split.

From these experiments, we see that our flocking-like technique has good performance on datasets *g241c* and *g241d*, has average performance on *Digit1*, and has poor performance on *USPS* and *BCI*. Actually, all the algorithms have poor performance on *BCI*. The best results from the literature have an error rate of 47.95 and 44.83% (10 and 100 initially labeled items, respectively). A simple blind guessing would result in an error around 50%, almost the same as obtained from all the algorithms.

Oddly, our technique results in error much above 50% for low values of β_1 on dataset *USPS*. We clearly see that the accuracy depends strongly on β_1, but even for higher values, poor results are still obtained. Actually, the dependency on the value of β_1 can be noticed on all the datasets.

Datasets *g241c*, *g241d*, and *Digit1* each follows, respectively, one of three assumptions as discussed by Chapelle et al. [1], namely *cluster*, *smoothness*, and *manifold* assumption. Hence, these experiments suggest that the proposed technique acts quite well for cluster and smoothness assumptions.

The k-NN technique is very fast, having a linear time complexity, assuming that the number of labeled items is much smaller than the unlabeled ones. However, it is so simple that it usually leads to worse classification, as shown if Fig. 4. Actually, it does not even take the unlabeled distribution into account. On the other hand, most of the other techniques have at least a quadratic time complexity. In particular, graph-based techniques usually employ an adjacency matrix, which ultimately leads to a quadratic complexity. As discussed in the complexity section, our technique has a sub-quadratic time complexity, so we achieve a balance between good classification accuracy and acceptable computational cost.

V. CONCLUDING REMARKS

We are aware that this flocking-like learning technique is a bit complicate, having many details and many parameters to be adjusted. However, we expect that such details can provide us more richness than difficulties. The decentralized, distributed interactions among the data items are quite robust. The rich set of variables of this dynamical system may provide us information about the dataset under analysis, such as class overlapping, wrong labeling, etc. Deeper studies over this model are in course, in order to get a better understanding of its potentialities and limitations. Those studies may also reveal simpler ways to set the model parameters, i.e., given a dataset, some procedure could be used to automatically set almost all the parameters.

Experiments on artificial and benchmark datasets gave an idea about the classification accuracy that can be achieved. Specially on artificial datasets, we see that this model is able to evidence regions under dispute between different classes (like a class overlapping). More than just propagating the labels from labeled to unlabeled items, this feature could also be used to perform soft labeling.

Despite the richness of this flocking-like learning model, we have also focused in keeping this model fast enough to work on large datasets. A great effort was given by developing and applying a space-partitioning technique, which enables a sub-quadratic time cost, although a deeper, more precise analysis is still needed.

ACKNOWLEDGMENT

This work is supported by The State of São Paulo Research Foundation (FAPESP) and by the Brazilian National Research Council (CNPq).

REFERENCES

[1] O. Chapelle, B. Schölkopf, and A. Zien, Eds., *Semi-Supervised Learning*. The MIT Press, 2006. [Online]. Available: http://www.kyb.tuebingen.mpg.de/ssl-book

[2] O. Chapelle, V. Sindhwani, and S. S. Keerthi, "Optimization techniques for semi-supervised support vector machines," *Journal of Machine Learning Research*, vol. 9, pp. 203–233, 2008.

[3] T. C. Silva and L. Zhao, "Network-based stochastic semisupervised learning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 23, no. 3, pp. 451–466, 2012, doi:10.1109/TNNLS.2011.2181413.

[4] T. H. Cupertino, R. Gueleri, and L. Zhao, "A semi-supervised classification technique based on interacting forces," *Neurocomputing*, vol. 127, pp. 43–51, 2014, doi:10.1016/j.neucom.2013.05.050.

[5] T. Vicsek, A. Czirók, E. Ben-Jacob, I. Cohen, and O. Shochet, "Novel type of phase transition in a system of self-driven particles," *Physical Review Letters*, vol. 75, no. 6, pp. 1226–1229, 1995.

[6] T. Vicsek and A. Zafeiris, "Collective motion," *Physics Reports*, vol. 517, pp. 71–140, 2012, doi:10.1016/j.physrep.2012.03.004.

[7] X. Cui, J. Gao, and T. E. Potok, "A flocking based algorithm for document clustering analysis," *Journal of Systems Architecture*, vol. 52, pp. 505–515, 2006, doi:10.1016/j.sysarc.2006.02.003.

[8] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," *ACM SIGGRAPH Computer Graphics*, vol. 21, no. 4, pp. 25–34, 1987, doi:10.1145/37402.37406.

[9] ——, "Steering behaviors for autonomous characters," web page, 1999. [Online]. Available: http://www.red3d.com/cwr/steer/gdc99/

[10] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, pp. 21–27, 1967.

[11] J. Sun, S. Boyd, L. Xiao, and P. Diaconis, "The fastest mixing markov process on a graph and a connection to a maximum variance unfolding problem," *SIAM Review*, vol. 48, pp. 681–699, 2006.

[12] M. Belkin and P. Niyogi, "Laplacian eigenmaps for dimensionality reduction and data representation," *Neural Computation*, vol. 15, pp. 1373–1396, 2003.

[13] M. Belkin, P. Niyogi, and V. Sindhwani, "Manifold regularization: A geometric framework for learning from labeled and unlabeled examples," *Journal of Machine Learning Research*, vol. 7, pp. 2399–2434, 2006.

[14] M. Belkin and P. Niyogi, "Convergence of laplacian eigenmaps," *preprint, short version NIPS 2008*, 2008.

[15] D. S. Rosenberg, V. Sindhwani, P. L. Bartlett, and P. Niyogi, "A kernel for semi-supervised learning with multi-view point cloud regularization," *IEEE Signal Processing Magazine*, 2009.

[16] H. Q. Minh and V. Sindhwani, "Vector-valued manifold regularization," in *Proceedings of the 28th International Conference on Machine Learning*, 2011.

[17] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, pp. 273–297, 1995.

[18] O. Chapelle and A. Zien, "Semi-supervised classification by low density separation," in *Proceedings of the 10th International Workshop on Artificial Intelligence and Statistics*, 2005, pp. 57–64.

[19] T. Joachims, *Semi-Supervised Learning*. The MIT Press, 2006, ch. Transductive Support Vector Machines.

[20] ——, "Transductive learning via spectral graph partitioning," in *Proceedings of the International Conference on Machine Learning*, 2003, pp. 290–297.

[21] V. Sindhwani, P. Niyogi, and M. Belkin, "Beyond the point cloud: from transductive to semi-supervised learning," in *Proceedings of the 22nd International Conference on Machine Learning*, 2005, pp. 824–831.