Dynamic Self-Organising Swarm for Unsupervised Prototype Generation

Su Nguyen, Binh Tran, and Damminda Alahakoon
Centre for Data Analytics and Cognition, La Trobe University, Australia
Email: {p.nguyen4,b.tran,d.alahakoon}@latrobe.edu.au

***Abstract*—Growing big data has posed a great challenge for machine learning algorithms. To cope with big data, the algorithm has to be both efficient and accurate. Although evolutionary computation has been successfully applied to many complex machine learning tasks, its ability to handle big data is limited. In this paper, we proposed a dynamic self-organising swarm algorithm to learn an effective set of prototypes for big high-dimensional datasets in an unsupervised manner. The novelties of this new algorithm are the energy-based fitness function, the adaptive topological neighbourhood, the growing/shrinking capability, and the efficient learning scheme. Experiments with well-known datasets show that the proposed algorithm can maintain a very compact set of prototypes and achieve competitive predictive performance as compared to other algorithms in the literature. The analyses also show that prototypes generated by the proposed algorithms have a stronger separatability compared to those from other prototype generation algorithms.**

I. Introduction

In the last decade, with the emergence of the Internet of Things, sensor networks and social media, big data and its applications have gained more and more attention from both academia and industry. However, maintaining this huge amount of continuously growing data, requires an enormous storage capacity. In addition, it is challenging to use conventional machine learning (ML) algorithms to efficiently learn and extract useful patterns from such big data. Among different strategies proposed to cope with these challenges is to determine a representative subset of the original data.

Prototype selection (PS) and generation (PG) are data reduction techniques that have been shown effective in assisting ML algorithms scale well with datasets having a large number of instances or samples. By creating a representative dataset that is significantly smaller than the original one, these two techniques not only reduce the storage requirements, the computational complexity of the ML algorithms, but also enhance their noise tolerance to achieve a better classification accuracy. While prototypes returned by PS are chosen from the original instances, those returned by PG are new artificial instances generated based on the existing data. Both techniques have been studied for decades, resulting in a countless number of algorithms proposed in the literature [1]. However, to the extent of our knowledge, most proposed algorithms are supervised and require class labels for learning prototypes. This limitation makes these algorithms less attractive in real-world scenarios in which labelling big data is time-consuming and expensive.

In the unsupervised learning scheme, self-organising map (SOM) algorithm proposed by [2] can be used as a PG method in which SOM feature map captures the distribution of the data inputs and each neuron or node of SOM is a generated prototype. To train SOM, each instance will be used as the input signal and the neuron (or specifically its weights) that best matches the signal and its neighbours will be updated to be more similar to the signal. Although SOM has shown to be effective in data reduction and dimensionality reduction, choosing an optimal size for the grid requires domain expert or extensive trials. To address this problem, Alahakoon et al. [3] proposed the growing SOM (GSOM) method in which the number of neurons is automatically grown when the error of the existing neuron exceeds a threshold, which helps GSOM capture the data distribution more effective as compared to SOM. Nevertheless, since a predefined topological neighbourhood is needed to update the neurons' weights, it is difficult to efficiently train GSOM or modify GSOM as the data distribution changes. Growing neural gas (GNG) and Self-Organising Incremental Neural Network (SOINN) are other variants of SOM with growing capability. Different from SOM and GSOM, GNG and SOINN can self-configure topological neighbourhood based on the input data, which allows them to learn new patterns more quickly. However, similar to SOM and GSOM, only topological neighbourhood and input signal are used to update the weights or prototypes. Thus, there is a good chance that they will generate many redundant prototypes, i.e. inactive neurons learnt during the training process. Although SOINN and GSOM have a mechanism to review and evaluate the contribution of each prototype, it is only triggered periodically and in a centralised manner rather than incorporating this information directly into the learning step to improve the training efficiency.

Imitating the social interactions observed in bird flocking, Particle swarm optimisation (PSO) [4] has been proven an efficient approach to dealing with difficult optimisation problems such as feature selection [5] and clustering [6]. Among popular population-based algorithms, PSO is well-known with its simplicity and fast-convergence ability. A PSO algorithm for PG called Michigan PSO (MPSO) was proposed in [7] where each particle represents a single prototype and the whole swarm constructs a single solution. Prototypes evolved by MPSO have shown to be representative for the training data. However, MPSO uses supervised learning which confines its application to labelled data. Furthermore, although

978-1-7281-6929-3/20/$31.00 ©2020 IEEE

particles can be removed during the evolutionary process, MPSO still requires a predefined number of particles to start with, which is hard to define, especially in the context of big high-dimensional data. For unsupervised learning, O'Niell and Brabazon [8] proposed a self-organizing swarm (SOSwarm) algorithm to learn a low dimensional mapping of the original input data. In SOSwarm, learning step is governed by the particle velocity rather than the updating scheme used in SOM. Similar to SOM, SOSwarm does not possess a growing capability and the swarm size and topological neighbourhood still need to be predefined. In this study, we propose a new PSO algorithm called dynamic self-organising swarm (DSOS) that can automatically determine the appropriate number of prototypes from the unlabelled and big high-dimensional data efficiently. The novelty of this algorithm is a new learning scheme mimicking the foraging behaviours of animals to maintain the diversity and identify the efficient size of the swarm. Specifically, the contributions of this study are:

- The new unsupervised PSO-based approach to PG that can incrementally learn the distribution of input data.
- A new energy-based fitness function to evaluate the contribution of each particle/prototype to the whole solution in the context of unsupervised learning.
- A new technique to dynamically adjust the swarm size so that particles can be created and removed according to the incoming data.
- A new technique to adaptively define the swarm topology.
- A new learning scheme based on particles' energy to efficiently learn the distribution of incoming data.

II. Related Work

This section briefly reviews the related self-organising algorithms and PSO algorithms for PG in the literature.

A. *Self-organising algorithms for unsupervised learning*

There are many algorithms proposed in the literature to determine the optimal data representation of the original dataset. One of the most popular algorithms is SOM [9], an unsupervised artificial neural network (ANN) whose goal is to transform an original complex high-dimensional data into a low-dimensional (typically two-dimensional) space by preserving the topological relationships in the data. Different from other ANNs with backpropagation and gradient descent usually used for supervised tasks, SOM uses competitive learning and a topological neighbourhood to learn the low-dimensional representation of the input data. In SOM, the size of the map with a fixed number of nodes/neurons (usually arranged in a regular hexagonal or rectangular grid) needs to be predefined. First, the weights of each node are randomly initialised. During the training process, SOM is fed with a data instance (or signal) and the best match unit (BMU) or the winning node is identified, i.e. the node whose weights are most similar to those of the input instance. Then the weights of BMU and its neighbours will be updated to be more similar to the input. The neighbours are usually determined based on the grid-distance to the BMU. In the simplest version, the grid-distance to the BMU of neighbours is one. However, the neighbourhood is usually set such that the number of neighbours will decrease over time.

Growing SOM (GSOM) [3] is a variant of SOM which starts with the minimum number of nodes and incrementally grow as needed. With the growing capability, GSOM eliminates the need to predefine the size of the map (e.g. number of nodes). In GSOM, a spread factor is introduced to control the growth of the map as data is fed into the algorithm. By adjusting the spread factor, the user can change the granularity of the representation. Similar to GSOM, growing neural gas (GNG) [10] is a graph-based algorithm that can grow gradually to adapt to the data distribution. Within GNG, the connections of nodes, which defines the topological neighbourhood, are generated based on the distances between existing nodes and the input instance. The two nodes are connected or become neighbours if they are the BMU and the second best-matched node. The connections are gradually weaken or removed if the nodes connected with BMU are rarely realised as neighbours (i.e. second best node) over time.

The extended version of GNG with the utility criterion (GNG-U) [11] is also introduced to cope with non-stationary data. In this version, a node can be removed from the graph if its utility value falls below a threshold as compared to the accumulated error of the unit with the maximum error. This mechanism helps GNG-U track the distribution changes better than GNG and SOM. SOINN [12] is another graph-based algorithm similar to GNG in which a new learning scheme is introduced to efficiently and adaptively learn data representation. In addition, SOINN has a mechanism to periodically review and remove redundant nodes if necessary.

B. *Particle Swarm Optimisation*

Proposed by Kennedy and Eberhart in 1995, Particle Swarm Optimisation (PSO) [4] is an evolutionary computation algorithm inspired by the social behaviours of birds flocking. In PSO, a swarm of particles moves in the solution space. Traditionally, each particle has a position representing a candidate solution and a velocity showing the magnitude and direction of its move. During the evolutionary process, particles share the best solution they have explored so far (i.e. the personal best - $pbest$) using a specific communication topology. If they are fully connected, they learn the global best solution ($gbest$); otherwise, a local best $lbest$ is learnt. Using this information, they adjust their velocities to move towards fruitful areas to search for better solutions.

PSO has been shown effective in solving complex optimisation problems such as optimising the structures of fuzzy models [13] and detecting salient objects [14]. It has also been applied to PG [15]; however, PSO-based PG approach is still not popular. This may be due to the high computational cost of using the traditional PSO encoding scheme in evolving a population of candidate solutions, each of which is a set of prototypes. Maintaining a set of such solutions is prohibitively expensive in terms of memory and computational time especially with high-dimensional data. Therefore, the number of evolved prototypes and the population size are usually fixed and predefined as small values [15]. To address this problem, Cervantes et al. [7] proposed a new PSO approach called Michigan PSO (MPSO) which encodes a single solution in the whole swarm of particles, each of which represents a prototype. As in [15], each particle in MPSO is evaluated based on its prediction performance on the associate training instances. MPSO has shown effective and efficient in evolving representative prototypes. Nonetheless, as a supervised learning method, MPSO limits its applications to labelled data.

Self-organising swarm (SOSwarm) [8] is an unsupervised learning algorithm proposed to learn a low dimensional mapping of the original data. Inspired by SOM, SOSwarm represents each particle as a node in SOM. A fixed-size grid is used as a communication topology between particles. This means that whenever a particle is updated, the eight neighbouring particles will also be updated. Different from traditional PSO where $gbest$ is the fittest particle, $gbest$ in SOSwarm is set as the data input at the beginning of each iteration. SOSwarm then finds the firing particle which is the closest particle to $gbest$. The positions of the firing particle and its neighbours will be updated using the traditional PSO updating mechanism. Compared with SOM on ten UCI datasets, SOSwarm has shown competitive results in evolving representative prototypes. However, the number and mapping layout of the evolved prototypes in SOSwarm have to be predefined as in SOM.

III. Proposed Method

DSOS is a particle swarm algorithm that mimics foraging behaviours of animals. In DSOS, particles move and seek for food sources in a high-dimensional space. In this case, each data input is a food signal which attracts the particles to move to its proximity. A particle gains more energy when it is closer to the food source but its energy will drain over time. However, a food source is limited and only the particles closest to the food source can consume the food to gain energy. Particles that share a food source will be connected (become neighbours) and guide each other to move towards incoming food signals. If a food signal is too far away from existing particles, a new particle is created at this food source. If two connected particles do not share a food source, their connections will be gradually weakened and removed by an increasing repulsive force. Over time, particles with zero energy or no connection will be removed from the swarm. By modelling this swarm behaviour, we can efficiently and dynamically capture the data distribution. The rest of this section will present the detailed description of DSOS algorithm.

A. Representation

DSOS maintains a dynamic population $\mathcal{S}$ in which prototypes or particles are created and removed during the evolutionary process depending on their fitness and the data inputs. A prototype is encoded in the position of a particle that also has a velocity showing the direction and magnitude of its move. Both position and velocity of a particle are n-dimension vectors, given n as the dimensionality of the data. Each dimension corresponds to one feature of the original data. The value in each dimension of the particle position ranges from 0 to 1. This means that data is scaled to the range of $[0 \dots 1]$ before applying DSOS. Velocity values are also kept in this range. How particle position and velocity are updated will be presented in Section III-E.

B. Fitness function

Since each particle partly represents the original dataset and no data label is used, it is not straightforward to evaluate how good a single particle or prototype is. In this algorithm, the fitness of a particle should reflect how well the particle responds to the food signal. Mimicking the animal foraging principle, the fitness or $Energy$ of a particle, ranging from 0 to 1, should be increased based on Eq. (1) when it is moving closer to a new food signal d_i presented at each iteration. In addition, the new food source is shared by all particles located within a radius governed by the spread factor SF. As shown in Eq. (1), the larger the SF, the smaller the amount of energy each particle in the region receives.

$$Energy_p = Energy_p + \frac{1 - Dist(p, d_i)}{1 + |\{q \in \mathcal{S} | Dist(q, d_i) < SF\}|} \tag{1}$$

where $Dist(p, d_i)$ is the distance between particle p and data input d_i. The equation shows that the energy of a particle is proportional to its proximity to the new data input. Any distance measure can be used for the $Dist$ function. DSOS uses the popular Euclidean distance measure and all distances are scaled to $[0 \dots 1]$. It is easy to see that the gained energy is smaller if SF is large, i.e. the food source is shared by a larger number of particles.

On the other hand, particles also have to consume energy to survive. As a consequence, its energy will decrease over time. Eq. (2) is used to subtract a small amount of energy that a particle spends in each iteration.

$$Energy_p = Energy_p - \frac{1}{\lambda} \tag{2}$$

where λ determines the number of iterations that a particle survives without approaching any food source. In other words, it affects the life duration of a particle. When a DSOS particle is created, it has a default energy level of 0.5. After created, if it has not gained energy from any food source (i.e. no matching data input), its life will be terminated after $\lambda/2$ iterations (e.g. 500 if λ is set to 1000). This energy modelling is proposed to ensure that particles positioning in an area with a large number of food sources, i.e. well representing the data distribution, will likely to maintain high energy. With a long expected life duration or higher λ, the swarm $\mathcal{S}$ will cover more food sources or the particles will capture the distribution at a lower level of granularity.

C. Swarm growing and shrinking mechanism

In order to adapt itself to the distribution of the input data, the swarm will dynamically grow and shrink during the evolutionary process. The growing mechanism has to be carefully designed so that particles are only created when the existing particles cannot cover an incoming data input, i.e. when the incoming data input is located outside the coverage of its best-matched particle.

The coverage area $Coverage_p$ of a particle p is defined based on its neighbourhood radius ($Radius$) which is the distance between itself and its furthest neighbour. $Radius$ is dynamically updated during the evolutionary process. In addition, the final solution of DSOS which is a set of prototypes should also reflect the structure or the distribution of the data. This means that more particles should be created in areas with more data inputs. Because the energy of a particle reflects how much food or data inputs it has ever approach, particles with higher energy should have a smaller coverage area. Therefore, the coverage of a particle is proportional to its $Radius$ and inversely proportional to its $Energy$ as shown in Eq. (3).

$$Coverage_p = \frac{Radius_p}{Energy_p + \epsilon} \tag{3}$$

where ϵ is a very small value (e.g. 10^{-5}) added to avoid $Radius_p$ being divided by zero when the particle's energy drops to zero.

As mentioned in Section III-B , the larger SF, the smaller the energy of a particle can get. Based on Eq. (3), it means that the larger the coverage area of the particle is, which reduces the chance of creating new particles. In this way, SF is the control factor of DSOS's swarm growth.

On the other hand, to deal with noise and particles positioning in no-longer-fruitful areas, a shrinking mechanism is applied every λ iterations. Specifically, DSOS removes particles with no or one neighbour. Furthermore, particles with two neighbours are removed when they are out of energy. This mechanism helps DSOS eliminate redundant particles and keep the swarm as compact as possible. According to Eq. (1), particles will be more competitive with a higher spread factor, which leads to more particles with zero energy and hence a smaller swarm. This again shows the impact of the spread factor SF to the swarm size.

D. Dynamic topological neighbourhood

In the conventional PSO algorithm where each particle represents a solution of the problem, knowledge or experience of a particle is generally helpful for all other particles in exploring better solutions. Therefore, particles can communicate using a fixed topology such as star, ring, mesh, etc. On the other hand, a DSOS particle only represents part of a solution. While its experience may be useful to some surrounding particles, it should not share with all others to maintain the swarm diversity. In addition, since particles are created and deleted during the evolutionary process, the fixed topology are inapplicable. DSOS requires a dynamic topology that is automatically evolved and changed during the evolutionary process. This population-level dynamics has been observed in biological systems. Studies of the collective behaviour of bird flocks and fish schools suggested that group members self-organise by a combination of attractive and avoidance behaviours to maintain their inter-individual spacing [16]. Imitating this behaviour, DSOS particles self-organise by attracting neighbouring particles towards the new resource while pushing each other using repulsive forces between them.

Specifically, each DSOS particle maintains its own set of neighbours whose experience is relevant to its learning process. When a particle is created, it has no neighbours. During the evolutionary process, two particles are first connected as neighbours if there is a data input falling into their coverage areas. When this happens, they maintain the strongest connection or the weakest repulsive force. During the evolutionary process, whenever a data input d_i matches a particle p but not its neighbour q, the repulsive force $F_{p,q}$ between the two particles will be increased an $\Delta_{F_{p,q}}$ amount calculated using Eq. (4).

$$\Delta_{F_{p,q}} = 1 + \left(\sum_{i|i \in (p \cup NB_p)} Energy_i \right) \times |NB_q| \times \frac{Dist(q, d_i)}{MeanDist} \tag{4}$$

where NB_p is the set of p's neighbours, $|NB_q|$ is the number of q's neighbours, and $MeanDist$ is the average distance between d_i and all neighbours of p:

$$MeanDist = \frac{1}{|NB_p|} \times \sum_{i|i \in NB_p} Dist(i, d_i) \tag{5}$$

This mechanism is developed to avoid the swarm from forming a too crowded cluster, which can be redundant and inefficient. With Eq. (4), the repulsive force will be stronger if the considered neighbourhood NB_p has a high total energy (i.e. particles have been already close to the food signals) and many neighbours (i.e. crowded), and particles are far from the food signal (i.e. suggesting that they should not be connected to the particle p). When $F_{p,q}$ exceeds a maximum level, p and q are disconnected or no longer be neighbours. This repulsive force modelling allows particles to dynamically adapt its neighbourhood to the nature of the incoming data and helps DSOS overcome the limitation of the fixed topological neighbourhood in SOM and SOSwarm. In terms of algorithm, the way DSOS establishes the connections between particles are similar to those of GNG and SOINN; however the novelty of DSOS is the repulsive force modelling to efficiently discriminate unrelated patterns.

E. Learning Mechanism

Different from the traditional PSO where all particles move at each iteration, a DSOS particle only moves when a new data input matches itself or one of its neighbours. This learning mechanism not only fit to the incremental learning approach but also avoid the high computation time of PSO, which is a common problem of evolutionary computation algorithms.

An iteration of DSOS starts when there is a new data input d_i. To see if the existing particles or prototypes can represent d_i, DSOS finds two best-matched particles to d. Two scenarios can happen. If d_i does not fall into both coverage areas of the two best matching particles, a new particle will be created at its location as mentioned in Section III-C. Otherwise, d_i is considered as the new local best of the best-matched particle p. In this case, p will move towards this new local best to better cover d_i. Eq. (6) and (7) describe how the position and velocity of p are updated.

$$Pos_p = Pos_p + Vel_p \tag{6}$$

$$Vel_p = \frac{1}{\#Matches} \times (d - Pos_p) \tag{7}$$

where Pos_p and Vel_p are the position/prototype and velocity of particle p, respectively. $\#Matches$ is the number of times particle p is identified as the best-matched particle. A large value of $\#Matches$ indicates that p can represent a large amount of data. Therefore, its moving step should be smaller, enabling p to converge into a representative prototype.

The information about the new local best d_i is also shared with p's neighbours to mimic the attraction behaviours of group members. Eq. (8) shows the velocity updating mechanism of a p's neighbouring particle q. The velocity Vel_q is updated in a similar way as p but with a much smaller moving step (divided by 100 [12]) to reflect an indirect effect of the local best d_i. Furthermore, Vel_q also depends on $Energy_q$ to allow particles with lower energy (i.e. not approaching much food) to move faster to the new food source d_i.

$$Vel_q = \frac{1 - Energy_q}{100 \times \#Matches} \times (d_i - Pos_p) \tag{8}$$

The overview of DSOS is presented in Algorithm 1. The swarm is initialised with two particles based on the first two incoming data inputs. For every λ iterations, the shrinking mechanism described in Section III-C is applied and the max_repul_force is heuristically recalculated based on the total energy of the whole swarm. As can be seen from Algorithm 1, DSOS can be terminated at any time to retrieve the current set of prototypes evolved so far. This incremental and on-line learning capability makes DSOS a suitable preprocessing method for big data.

Algorithm 1: DSOS Algorithm

```
Input  : Training data, $SF$ , $\lambda$
Output: Set of Prototypes
1  begin
2      Initialise 2 particles using the first 2 data inputs;
3      $max\_repul\_force = \infty$;
4      for each next data input $d_i$ do
5          $p_0 \leftarrow$ The closest particle to $d_i$ with distance $dist_0$;
6          $p_1 \leftarrow$ The second closest to $d_i$ with distance $dist_1$;
7          Calculate $Coverage_{p_0}$ and $Coverage_{p_1}$ based on Eq. (3);
8          if ($dist_0 > Coverage_{p_0}$) or ($dist_1 > Coverage_{p_1}$) then
9              Create a new particle at position $d_i$, energy 0.5, no
                 neighbours, and an infinitive Radius;
10         else
11             $\#Matches_{p_0}$ ++;
12             Connect $p_0$ and $p_1$ with repulsive force 0;
13             Update $p_0$ and its neighbours using Eq. (7), (6), (1),
                 (8);
14             Increase repulsive force between $p_0$ and its neighbours
                 using Eq. (4);
15             Disconnect $p_0$ with neighbours that have the repulsive
                 force exceed $max\_repul\_force$;
16         end
17         Update $Radius$ of $p_0$ and $p_1$ to their farthest neighbour;
18         Decrease $Energy$ of all particles using Eq. (2);
19         if $(i+1)\%\lambda$ then
20             Delete particles that has no or one neighbour;
21             Delete particles that has 2 neighbours and 0 energy;
22             $max\_repul\_force = (\sum_{all\ particles} Energy)/3$ ;
23         end
24     end
25     return Position of all particles;
26 end
```

TABLE I
DATASETS

Dataset	#Features	#Inst.	#Class	%Smallest class	%Largest class
Spambase [17]	228	4,601	2	39.4	60.6
Churn [18]	49	7,032	2	26.6	73.4
Mushroom [19]	113	8,124	2	48.2	51.8
ICU [20]	53	11,773	2	10.7	89.3
Adult [21]	36	32,561	2	24.1	75.9
Bank [22]	57	41,188	2	11.3	88.7

IV. Experimental Design

A. Datasets

To test the performance of the proposed algorithm, we use six datasets with different levels of difficulties. In Table I, the datasets are sorted in ascending order of the number of instances. As shown in the last two columns, most of these datasets are unbalanced data.

B. Experiment Configuration and Parameter Settings

To test the performance of DSOS in PG, we compare DSOS with two other unsupervised methods, namely SOM and SOINN. In order to show which method creates better representative prototypes, we compare the classification accuracy of 1-Nearest-Neighbour (1NN) when using prototypes generated from these methods to classify new instances. However, the created prototypes of all these three unsupervised methods do not have any label. Therefore, majority voting is used to assign the most common class label of the training data inputs associating with a prototype as its class label (note that each training instance associates only with the closest prototype). These labelled prototypes are then used as training instances for 1NN to classify the test data. The results of 1NN using the raw data are also compared with DSOS. Balanced accuracy [23] on the test set are reported for all the compared methods.

For each method on each dataset, we conduct 30 independent runs with different orders of input data. For each dataset, 5-fold cross-validation is used to split data into training and test set. Each method uses the training set to learn a set of prototypes whose performance is then tested using the test set. Therefore, we have 150 results (i.e. 5 folds x 30 runs) for each dataset. The Wilcoxon statistical test is used to confirm if the results are significantly different.

SOM grid is set to 30 x 30 as these datasets have a large

TABLE II
AVERAGE RESULTS OF 30 INDEPENDENT RUNS.

Dataset	Method	#Prototypes	Test balanced accuracy		
			Best	Avg (Std)	T
Spambase	Full	3,680	78.52		+
	SOM	900	80.25	79.46 (0.51)	+
	SOINN	431	81.97	80.30 (0.75)	=
	DSOS	156	81.72	79.97 (0.70)	
Churn	Full	5,626	65.14		+
	SOM	900	68.49	67.70 (0.42)	+
	SOINN	421	69.16	68.16 (0.47)	+
	DSOS	305	69.53	68.69 (0.48)	
Mushroom	Full	6,499	93.81		–
	SOM	900	96.28	95.17 (0.76)	–
	SOINN	382	94.19	91.04 (1.68)	–
	DSOS	339	92.24	88.22 (1.92)	
ICU	Full	9,418	65.03		–
	SOM	900	60.98	59.83 (0.46)	–
	SOINN	361	59.00	57.59 (0.73)	+
	DSOS	330	59.57	58.64 (0.54)	
Adult	Full	26,049	69.62		–
	SOM	900	71.43	70.34 (0.41)	–
	SOINN	235	69.61	68.45 (0.42)	+
	DSOS	365	70.13	69.26 (0.42)	
Bank	Full	32,950	45.21		+
	SOM	900	53.12	51.30 (0.69)	+
	SOINN	739	55.87	53.48 (1.07)	=
	DSOS	499	57.25	53.03 (1.41)	

number of instances. In DSOS, SF and λ are experimentally set to 0.2 and 1000. For SOINN , age_{max} is 50 as chosen in [24] and $\lambda = 1000$ similar to DSOS.

V. RESULTS AND DISCUSSION

Table II shows the results of 1NN using the original instances (*"Full"*), the prototypes generated by SOM, SOINN and DSOS. The third column shows the number of prototypes generated by each method except for *Full* where the number of raw instances in the training set is shown. For SOM, since the grid size is predefined as 30 x 30, 900 prototypes are generated for all datasets. The next two columns present the best, the average and standard deviation of the class-balanced accuracy obtained in the 30 runs. The last column T displays the Wilcoxon significance test results (with the significance level of 0.05) of DSOS over the corresponding method on the test set. "+" or "–" means the result of DSOS is significantly better or worse than the baseline methods, respectively, while "=" means they are similar in the Wilcoxon tests. In other words, the more "+", the better the proposed methods.

A. DSOS versus Full

As can be seen from Table II, 1NN obtained significantly better results when using DSOS prototypes than using the original full set of instances (*Full*) on three datasets, namely Spambase, Churn and Bank. For example in the Bank dataset, using 499 prototypes evolved by DSOS, 1NN improved its performance 8% on average and 12% in the best case, compared to using 32,950 raw instances in the original dataset. On the remaining three datasets, DSOS obtained up to 6.5% lower accuracy than *Full* on average. However, when using DSOS prototypes, the number of instances that 1NN has to work with is dramatically reduced to only 1% to 5% of the training set size for all the datasets. This means that 1NN saves 95% to 99% of the running time in classifying a new instance.

B. DSOS versus SOM

While the number of prototypes generated by SOM is fixed to 900 for all datasets, the number of prototypes evolved by DSOS ranges from 160 to 499. Using a much smaller set of prototypes from DSOS, 1NN still obtained significantly better prediction performance than using SOM prototypes on three datasets, namely Spambase, Churn and Bank. On the Bank dataset, DSOS obtained 4% higher accuracy than SOM, although the number of DSOS prototypes is only about half of SOM. On the remaining three datasets, DSOS obtained 1% to 7% lower accuracy than SOM; however, with about two third fewer prototypes.

C. DSOS versus SOINN

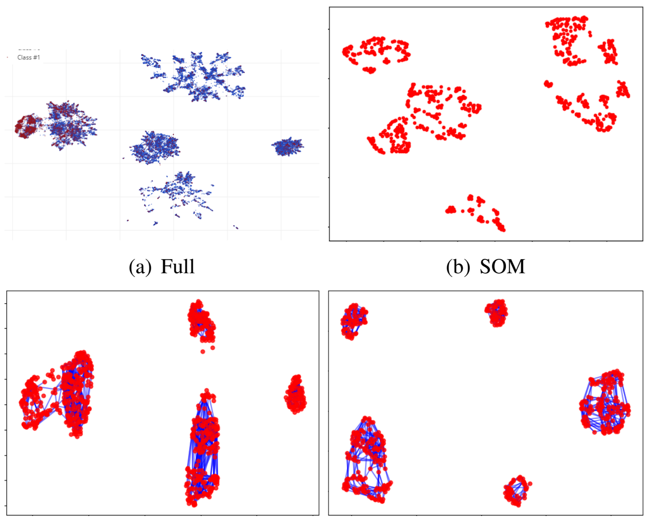
As can be seen in the third column of Table II, SOINN generates a larger number of prototypes than DSOS on all datasets except for the Adult dataset. In terms of prediction performance, DSOS obtained a significantly better accuracy than SOINN on three datasets, similar on two and worse on the remaining one. Although the accuracy difference between the two methods is less than 1%, SOINN generates up to 275 more prototypes than DSOS on almost all datasets. For example, on the largest dataset (Bank), although both have similar average accuracy, DSOS obtained 1.4% accuracy higher than SOINN in the best case while generated 240 fewer prototypes than SOINN. This has shown that DSOS performs a better search than SOINN.

It is noticed that the number of prototypes generated by DSOS is always proportional to the size of the dataset, ranging from 156 for the smallest dataset to 499 for the largest one. Since DSOS uses an incremental learning approach as described in Section III, the dataset size is not available to the algorithm. Therefore, the ability to grow its swarm proportionally to the dataset size reflects the effectiveness of DSOS in automatically capturing the complexity of the data. This may explain why DSOS maintains its performance on large datasets. On the other hand, this phenomenon is not seen in SOINN. For example, while 431 prototypes are generated by SOINN for Spambase with 3680 instances, only 235 are generated for Adult with 26049 instances.

In summary, among 18 comparisons with the three baseline methods, DSOS won 9 cases, drew 2 and lost 7. However, in terms of data reduction, DSOS evolved the smallest set of prototypes on five out of six datasets. The results showed that DSOS can generate representative prototypes to the original data and is more effective than the compared PG methods.

D. Visualising Generated Prototypes

In order to see how DSOS generated prototypes are different from other methods, we use UMAP [25] to visualise the



(a) Full (b) SOM (c) SOINN (d) DSOS

Fig. 1. Generated Prototypes on Bank.

generated prototypes of all the three PG methods along with the original data. UMAP is a recently proposed machine learning algorithm to obtain a low dimensional representation (two dimensions in this paper) of the raw data set. Becht et al. [26] show that UMAP performs exceptionally well across all investigated aspects from preserving the global structure, robustness, and running times as compared to the state-of-the-art algorithms such as t-SNE [27] and Autoencoder SCVIS [28]. The best result of each method on the first training fold of the largest dataset (Bank) is chosen to visualise.

Fig. 1 shows the visualisations of the original instances in (a) and the prototypes generated by SOM (b), SOINN (c) and DSOS (d) for the bank dataset. For Fig. 1 (b), (c), and (d), each red node represents a generated prototype. The blue lines in Fig. 1 (c) and (d) display the connections between prototypes generated by SOINN and DSOS. Based on the visualisation of the full dataset, it is easy to see that there are five main clusters. It is a bit tricky to identify the clusters in SOM's visualisation as generated prototypes are not well separated (especially ones at the top right corner of Fig. 1(b). In Fig. 1(c), SOINN can separate the prototypes slightly better than SOM but there is still a lot of confusion, especially with the bottom and left clusters. For this example, only DSOS shows five separated clusters as observed with the full dataset.

To have a more intuitive visualisation of the generated prototypes, we run the three methods on the MNIST dataset [29] which has 70,000 images of numbers 0 to 9. Each image has 784 features (28x28 pixels). Fig. 2 shows the UMAP visualisation of the generated prototypes on MNIST and the 1NN accuracy on the test set. It is clear that the prototypes generated by SOM in Fig. 2(a) cannot separate classes effectively. Except for straightforward digits such as zeros, ones, and sixes, prototypes representing other digits are very confused in this visualisation. For example, different groups of sevens and nines are interleaved in the visualisation. SOINN, in Fig. 2(b), is slightly better than SOM in this case when clusters of twos and sevens can be identified more easily. DSOS presents the best performance for the MNIST dataset in this experiment. In Fig. 2(c), very clear clusters are formed for straightforward digits (zeros, ones, twos, sixes) and prototypes representing other more confusing digits (fours, sevens, nines and threes, fives, eights) are also nicely separated. This result is very encouraging as DSOS learns the prototypes in a pure unsupervised learning manner. By generating a good representative prototypes, DSOS can help 1NN achieves better accuracy on classification task as compared to SOM and SOINN.

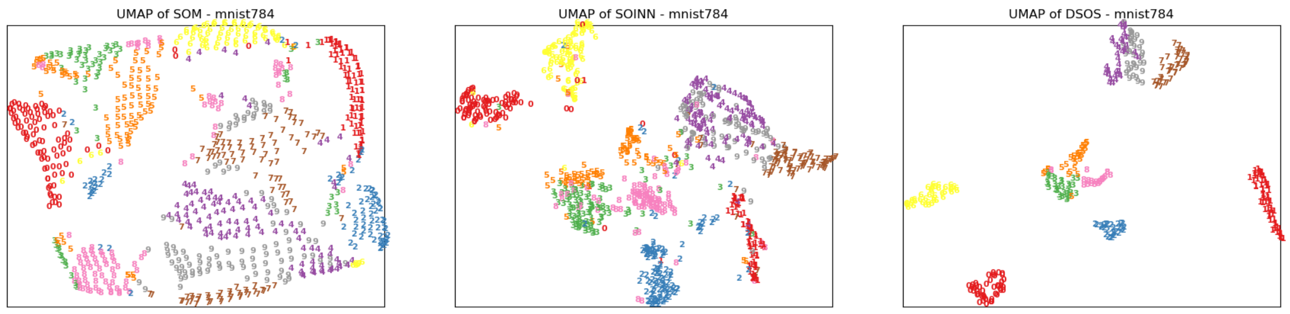
E. DSOS Evolutionary Process

To better explain the performance of DSOS, we investigate DSOS evolutionary process. Figure 3 shows the evolutionary process of one DSOS run on Bank. In Figure 3(a), the red and green lines display the swarm size and total energy of the whole swarm, respectively. As expected, the swarm size gradually grows over time and have a big drop every $\lambda = 1000$ iterations. However, the maximum swarm size stops increase after the $20,000^{th}$ iterations. This suggests that the swarm size is well control and the swarm only needs to grow if new patterns emerge. However, it is noted that the total energy shows a much smaller fluctuation as compared to the swarm size. The observation indicates that DSOS mainly remove unwanted/redundant prototypes, i.e. isolated or zero-energy particles; and therefore, a set of representative prototypes are still well-preserved. In Figure 3(b), the moving average error (differences between inputs and the best-matched particles) is averaged over the last 5000 iterations. It is easy to see that the errors decrease as the swarm adapts to the data inputs.

VI. Conclusion

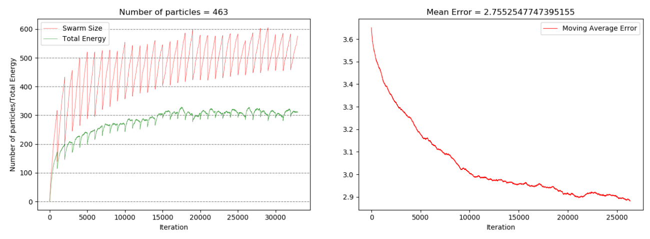
Handling big high-dimensional data is challenging for evolutionary and swarm algorithms. To deal with this challenge, this paper introduces dynamic self-organising swarm, an efficient algorithm based on the foraging behaviours of animals to incrementally learn the distribution of input data by determining the set of representative prototypes from the original dataset. The novelties of this algorithm is an energy-based fitness function, mechanisms to adjust the swarm size and the topological neighbourhood of the particles, and a new learning scheme. Experiments with popular datasets in the literature show that the proposed algorithm is very competitive as compared to well-established algorithms in the literature. Further analyses reveal that the proposed algorithm can generate prototypes with a strong separability of classes although it is a purely unsupervised learning algorithm.

This study is just a preliminary step to demonstrate the applicability of swarm algorithms in analysing and extracting patterns from big high-dimensional data. The proposed algorithm is designed by modelling the foraging behaviours of animals but it can be applied naturally to cope with complex machine learning tasks. In future studies, more analyses are



(a) SOM (1NN-$accuracy = 87.5\%$) (b) SOINN (1NN-$accuracy = 87.85\%$) (c) DSOS (1NN-$accuracy = 90.03\%$)

Fig. 2. Generated Prototypes on MNIST and 1NN's accuracy on the test set.



(a) Swarm size and total energy (b) Moving average error

Fig. 3. DSOS evolutionary process on Bank.

needed to fully understand the behaviours of this new algorithm as well as identify opportunities to enhance its efficiency.

References

[1] S.-W. Kim and B. J. Oommen, "A brief taxonomy and ranking of creative prototype reduction schemes," *Pattern Analysis & Applications*, vol. 6, no. 3, pp. 232–244, 2003.

[2] T. Kohonen and S.-O. Maps, "Springer series in information sciences, vol. 30," 1995.

[3] D. Alahakoon, S. K. Halgamuge, and B. Srinivasan, "Dynamic self-organizing maps with controlled growth for knowledge discovery," *IEEE Transactions on neural networks*, vol. 11, no. 3, pp. 601–614, 2000.

[4] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *Proceedings of the 6th International Symposium on Micro Machine and Human Science*, 1995, pp. 39–43.

[5] B. Tran, B. Xue, and M. Zhang, "Variable-length particle swarm optimisation for feature selection on high-dimensional classification," *IEEE Transactions on Evolutionary Computation*, 2018.

[6] D. O'Neill, A. Lensen, B. Xue, and M. Zhang, "Particle swarm optimisation for feature selection and weighting in high-dimensional clustering," 2018, pp. 1–8.

[7] A. Cervantes, I. Galván, and P. Isasi, "Michigan particle swarm optimization for prototype reduction in classification problems," *New Generation Computing*, vol. 27, no. 3, pp. 239–257, 2009.

[8] M. O'Neill and A. Brabazon, "Self-organising swarm (SOSwarm)," *Soft Computing*, vol. 12, no. 11, pp. 1073–1080, Sep 2008.

[9] T. Kohonen, "Self-organized formation of topologically correct feature maps," *Biological Cybernetics*, vol. 43, no. 1, pp. 59–69, Jan. 1982.

[10] B. Fritzke, "A growing neural gas network learns topologies," in *Proceedings of the 7th International Conference on Neural Information Processing Systems*, ser. NIPS'94, 1994, pp. 625–632.

[11] ——, "A self-organizing network that can follow non-stationary distributions," in *Proceeedings of International Conference on Artificial Neural Networks: ICANN'97*. Springer Berlin Heidelberg, 1997, pp. 613–618.

[12] S. Furao and O. Hasegawa, "An incremental network for on-line unsupervised classification and topology learning," *Neural Networks*, vol. 19, no. 1, pp. 90–106, 2006.

[13] K. Y. Chan, T. S. Dillon, and C. K. Kwong, "Modeling of a liquid epoxy molding process using a particle swarm optimization-based fuzzy regression approach," *IEEE Transactions on Industrial Informatics*, vol. 7, no. 1, pp. 148–158, 2011.

[14] S. Afzali, B. Xue, H. Al-Sahaf, and M. Zhang, "A supervised feature weighting method for salient object detection using PSO," in *Proceedings of the 2017 IEEE Symposium Series on Computational Intelligence*. IEEE, 2017, pp. 1–8.

[15] L. Nanni and A. Lumini, "Particle swarm optimization for prototype reduction," *Neurocomputing*, vol. 72, no. 4, pp. 1092 – 1097, 2009.

[16] I. D. Couzin and J. Krause, "Self-organization and collective behavior in vertebrates," in *Advances in the Study of Behavior*. Academic Press, 2003, vol. 32, pp. 1 – 75.

[17] L. F. Cranor and B. A. LaMacchia, "Spam!" *Commun. ACM*, vol. 41, no. 8, pp. 74–83, Aug. 1998.

[18] Kaggle, *Telco Customer Churn*, 2019 (accessed November 11, 2019). [Online]. Available: https://www.kaggle.com/blastchar/telco-customer-churn

[19] J. C. Schlimmer, "Concept acquisition through representational adjustment," Ph.D. dissertation, 1987.

[20] A. E. Johnson, J. Aboab, J. Raffa, T. Pollard, R. Deliberato, L. Celi, and D. Stone, "A comparative analysis of sepsis identification methods in an electronic database," *Critical Care Medicine*, vol. 46, no. 4, p. 494–499, 2018.

[21] R. Kohavi, "Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, ser. KDD'96. AAAI Press, 1996, pp. 202–207.

[22] S. Moro, P. Cortez, and P. Rita, "A data-driven approach to predict the success of bank telemarketing," *Decision Support Systems*, vol. 62, pp. 22–31, 2014.

[23] U. Bhowan, M. Johnston, M. Zhang, and X. Yao, "Reusing genetic programming for ensemble selection in classification of unbalanced data," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 6, pp. 893–908, Dec 2014.

[24] F. Shen and O. Hasegawa, "A fast nearest neighbor classifier based on self-organizing incremental neural network," *Neural Netw.*, vol. 21, no. 10, p. 1537–1547, Dec. 2008.

[25] L. McInnes, J. Healy, N. Saul, and L. Großberger, "UMAP: uniform manifold approximation and projection," *J. Open Source Software*, vol. 3, no. 29, p. 861, 2018.

[26] E. Becht, L. McInnes, J. Healy, C.-A. Dutertre, I. W. H. Kwok, L. G. Ng, F. Ginhoux, and E. Newell, "Dimensionality reduction for visualizing single-cell data using umap," *Nature Biotechnology*, vol. 37, p. 38–44, 2019.

[27] L. van der Maaten and G. Hinton, "Visualizing data using t-SNE," *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 2008.

[28] J. Ding, A. Condon, and S. P. Shah, "Interpretable dimensionality reduction of single cell transcriptome data with deep generative models," in *Nature Communications*, 2018.

[29] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/