Cluster Computing
https://doi.org/10.1007/s10586-020-03093-3

Multi-level particle swarm optimisation and its parallel version for parameter optimisation of ensemble models: a case of sentiment polarity prediction

Gregorius Satia Budhi[1,2] · Raymond Chiong[1] · Sandeep Dhakal[1]

Received: 9 September 2019 / Revised: 8 January 2020 / Accepted: 9 March 2020

Abstract
Ensemble learning is increasingly used in sentiment analysis. Determining the parameter settings of ensemble models, however, is not easy. Besides its own parameters, an ensemble model has base-predictors that have their individual parameters. Some ensemble models use a specific base-predictor and could be optimised using standard metaheuristics such as the Particle Swarm Optimisation (PSO) approach. Optimising ensemble models with multiple base-predictor candidates is more complicated and challenging, as there are multiple options to choose from. We therefore propose Multi-Level PSO (ML-PSO) and Parallel ML-PSO (PML-PSO) to optimise the parameters of ensemble models, especially those with multiple base-predictors, for sentiment analysis. The idea is to utilise multiple PSOs as particles of the main PSO. The main PSO optimises ensemble-model parameters and determines the best base-predictor, whereas PSOs within it optimise the corresponding base-predictor's parameters. Experimental results using Bagging Predictors as the underlying ensemble model show that ML-PSO can improve prediction accuracy, while PML-PSO is able to speed up the processing time and further improve the accuracy.

Keywords Particle swarm optimisation · Parallelism · Machine learning · Sentiment analysis

1 Introduction

Sentiment polarity detection, or more generally known as sentiment analysis, is the process of automatically and systematically detecting the sentiment or opinion of a given text. In addition to feature selection, the outcome of sentiment analysis primarily depends on the detection algorithm applied [1–4]. The majority of methods used for sentiment analysis belong to the machine learning domain. These methods are usually applied to predict the sentiment polarity of social media texts, online product reviews or other kinds of texts [2–8]. Due to the extensive amount of online texts such as product reviews, tweets and other social media texts, a system capable of automated sentiment analysis is vital in the online environment [9, 10]. Analysis using machine learning generally begins with training the machines to make them capable of discriminating the texts. The accuracy of the prediction model is determined by the quality of this training process [4, 11], and also how features of the text are extracted [12, 13]. However, acquiring the correct parameter settings for machine learning models to obtain the desired accuracy is challenging [14, 15]. Researchers usually apply either the original set of parameters used by the authors of the algorithms, improved settings suggested by other researchers, or default settings of the software components. These approaches, however, often do not produce optimal results, since the parameter settings are not tuned to the problem at hand.

Metaheuristics and nature-inspired algorithms [16–18], such as swarm intelligence and evolutionary algorithms, are regularly applied to optimise machine learning models. Compared to other metaheuristic optimisation techniques, particle swarm optimisation (PSO) offers advantages such

✉ Raymond Chiong
Raymond.Chiong@newcastle.edu.au

[1] School of Electrical Engineering and Computing, The University of Newcastle, Callaghan, NSW 2308, Australia

[2] Informatics Department, Petra Christian University, Surabaya 60236, Indonesia

as easy implementation, rapid convergence, avoidance of local optima, and is also computationally inexpensive because of low memory and CPU speed requirements. It has, therefore, been widely used in many research areas, such as function optimisation [18, 19], fuel management optimisation [20], energy-efficient scheduling [21], malicious web domain identification [22, 23], image semantic classification [24], and so on. Some examples of the use of PSO for sentiment analysis include parameter optimisation [1, 5, 14, 15, 25, 26], feature selection [1, 6, 12, 15], and clustering [27], among others. Parallel processing power has also been applied to speed up the PSO process [20, 24, 28–30].

In this paper, we introduce two novel methods based on PSO to obtain best possible parameter settings for an ensemble of classifiers. The first method introduced, called Multi-Level PSO (ML-PSO), comprises two levels. The first level consists of a main PSO algorithm, while the second level consists of multiple PSOs (hereafter referred to as inner PSOs) representing the particles in the main PSO. Each inner PSO can optimise a base-predictor, and each classifier has different sets of parameters. The inner PSOs report their results to the main PSO. The main PSO then chooses the best classifier and the best possible setting of its parameters. Due to the hierarchical nature of ML-PSO, the computing resources required are multiplied by the number of particles in each inner PSO and the number of particles in the main PSO. Given that PSO algorithms need a large number of particles and iterations to achieve satisfactory results, the computing resources required by ML-PSO can be enormous. To overcome this problem, we propose the second method, called Parallel ML-PSO (PML-PSO), which applies parallelism to ML-PSO. In PML-PSO, each inner PSO is set as a small semi-independent module that can be run separately in a different thread or process (CPU) using a different set of resources, and in parallel with other inner PSOs, thus reducing the processing time. This approach also helps overcome the processing-time limit usually applied to grid computing facilities.

In our experiments, we use Bagging Predictors (BP) [31] as the underlying ensemble model, with classifiers including the Logistic Regression (LR) [32], Linear-kernel Support Vector Machine (LSVM) [33], and Multilayer Perceptron (MLP) [34] as base-predictor candidates for BP. These classifiers were chosen because they have performed well in previous studies on sentiment polarity prediction of review texts (e.g., see [3–8]). Some studies have shown that ensemble algorithms can further increase the accuracy and other measurements [2, 4, 35], and BP was identified as the best ensemble algorithm for the problem at hand in one of these previous studies [4]. We also found that BP was the most flexible ensemble model and can be paired with any single classifier as its base-predictor. This flexibility makes BP the most suitable ensemble-model candidate for our proposed multi-level approach. The task of optimising ensemble parameters, choosing the base-predictor from a number of candidates, and optimising the parameters of the base-predictor, can be considered a multi-level problem, which needs to be solved in a multi-level manner. Our experimental results show that ML-PSO can improve sentiment polarity prediction by more than 14% compared to BP using a default set of parameters and the default base-predictors. Furthermore, PML-PSO is able to speed up the processing time and enhance the performance of BP following training with a large amount of data.

The rest of this paper is organised as follows. In the next section, the related literature background on PSO for sentiment analysis is briefly reviewed. The design of ML-PSO and PML-PSO is described in detail after that, followed by experimental results and discussions about their performance. Finally, we draw conclusion and highlight future research directions.

2 Background

PSO, inspired by animal swarming behaviour such as bird flocking or fish schooling, was originally developed by Kennedy and Eberhart in 1995 [36]. Different variants of the original PSO have been proposed by researchers over the years to further improve the process or make it more suitable for a particular problem. Examples include the discrete binary PSO (BPSO) [37], accelerated PSO [38], hybrid PSO with genetic operators [21] or new formula addition [24, 39], and ensemble of multiple PSOs [40].

PSO has also been widely used in sentiment analysis. Basari et al. [14] used a Support Vector Machine (SVM) to detect the polarity of Twitter's movie reviews, with its parameters optimised by a PSO algorithm. Similarly, Li et al. [1] used PSO to optimise the parameters of their SVM, to detect the sentiment polarities of texts from a Chinese-based social media website with user-generated content. In addition, they used PSO to optimise feature dimensions used for SVM training. Wahyudi and Kristiyanti [41] utilised PSO for feature selection and combined it with an SVM to classify smartphone product reviews.

Two types of modified BPSO were proposed by Shang et al. [13] to select features for Chinese text sentiment classification using three types of machine learning algorithms (i.e., SVM, Naïve Bayes (NB), and CART4.5). Rapid PSO-based feature selection, which is similar to accelerated PSO [38], was proposed by Kumar and Kumar [42] for sentiment analysis. A combination of PSO and a genetic algorithm (GA), named PSO-GA [43], was proposed by Sonagi and Gore [12] to be implemented for feature selection in sentiment analysis using an SVM classifier.

Mikula and Machova implemented PSO and bare-bones PSO to create annotations for a dictionary used for sentiment analysis in Slovak [44]. Meanwhile, Bansal and Kaur [6] compared the performances of ant colony optimisation and PSO for the optimisation of feature selection from tweets; they used the optimised features in NB and SVM classifiers to predict the sentiment polarity of the data. More recently, Budhi et al. [26] proposed a multi-PSO model to select the best classifier for sentiment analysis and optimise the parameters of the selected classifier.

3 Methods

3.1 Particle swarm optimisation

We use basic PSO as the foundation of our methods [21]. It can be implemented in a few lines of code and uses only primitive mathematical operators. The PSO algorithm used in our study is described as follows:

Algorithm 1 Basic PSO

```
1. Initialise the particles' positions and velocities randomly. Initialise each particle's personal best (Pb) and the swarm's
   global best (Gb);
2. Update the particles' positions using Eq. 1 and Eq. 2:
3.     $v_{lq}^{t} = wv_{lq}^{t-1} + c_1 rnd_{11}(Pb_{lq}^{t-1}\text{-}U_{lq}^{t-1}) + c_2 rnd_{12}(Gb_{lq}^{t-1}\text{-}U_{lq}^{t-1})$     (1)
4.     $U_{lq}^{t} = U_{lq}^{t-1} + min\{Vmax, v_{lq}^{t}\}$     (2)
5. Update Pb and Gb;
6. Update the inertia-weight, w, randomly;
7. If terminating criteria are met, stop and report Gb; else go to step 2.
```

In Algorithm 1, U_{lq}^{t}, Pb_{lq}^{t}, Gb_{lq}^{t} and v_{lq}^{t} represent the current position, personal best position *Pb*, global best position *Gb*, and the velocity of particle *l* at dimension *q* and iteration *t*, respectively; rnd_{11} and rnd_{12} are random values from 0 to 1; c_1 and c_2 are the weights for regulating the influence of *Pb* and *Gb*; *w* is the inertia weight for balancing personal and global exploration abilities of the swarm, and *Vmax* is the maximum velocity to stop the particle from moving beyond its limitation [21].

3.2 Classifiers

While the proposed method can apply any classifier for sentiment analysis, in our current study, we opt for the three best classifiers identified by prior experiments [4], namely the LR, SVM, and MLP. We also investigate an ensemble of classifier algorithms, using BP, which was identified as the best ensemble algorithm for the problem at hand [4], as the underlying model. BP uses several single predictors to build a cluster of predictors, and is commonly adopted in many areas [35, 45]. These predictors are trained through a bootstrapping process that replicates the training set. BP predicts a class using plurality voting [31]. In addition to the three classifiers mentioned above, we also include Classification And Regression Trees (CART) and Nearest Neighbour (NNb) algorithms, since they are the default base-predictors for BP [31].

LR is a member of the generalised linear model family created by Nelder and Wedderburn in 1972 [46], and improved by Hastie and Tibshirani in 1990 [47]. Traditional linear models are limited to using continuous and normally distributed variables, which is not always desirable. The generalised linear models overcome this problem by using non-normal dependent variables [48, 49]. In LR analysis, the dependent variables can either be unordered polytomous (polytomous nominal) or ordered polytomous (polytomous ordinal); while the independent variables (predictors) can either be interval/ratio variables or dummy variables for representing a limited number of categories [32].

The SVM learns from a training dataset and generalises for correct predictions on unseen data. It works by separating a hyperplane and maximises the separation distance. Larger the margin, lower the error generated by the classifier [33]. SVMs are widely used in many research areas [50–55]. In this study, we consider the LSVM, as previous studies have found it to perform better than other types of kernels (e.g., see [4]).

The MLP is a feedforward artificial neural network normally used as a supervised model for pattern recognition and classification [56]. This model minimises the error in its results by computing the weights in its network. The algorithm continually updates the weights to achieve the best configuration and consists of two phases: feed-forward and backpropagation. In the feed-forward phase, training data is forwarded to produce an output, then the difference between the real output and desired target is calculated to produce an error. This error is then used to update the weights accordingly [34]. This algorithm has been used and improved by researchers in different areas [22, 57–62].

In the original BP ensemble, CART and NNb algorithms were used as the base-predictors [31]. CART or the Decision Tree classifier was developed by Quinlan [63] based on Hunt's algorithm [64]. As the name suggests, it is a tree-like model, creating decision trees for classification and prediction purposes. This classifier is a useful explanatory tool for expressing the cause and effect chain [65]. It has been used for text classification [66, 67] and many other applications [68, 69]. This algorithm is typically used as a base-predictor for ensemble methods [31, 70, 71], and is also widely used for solving classification and regression problems. Similarly, the NNb is a long-established algorithm that is often used to estimate an unknown sample using the closest instances [72]. This algorithm is still widely used [22] and regularly improved [73].

3.3 Multi-level PSO

The proposed method, ML-PSO, can be seen in Fig. 1. By design, this method consists of two levels. The first level has a main PSO, whose particles are PSOs themselves (called inner PSOs). We use basic PSO from Kennedy and Eberhart [36] for both the main and inner PSOs. However, they can be easily replaced by other PSO variants if required. It is also possible to create an ensemble of PSOs from different variants of PSO algorithms as suggested by Lynn and Suganthan [40]. The inner PSOs (particles of the main PSO) work independently of each other and report their results to the main PSO for the adjustment of its *Pb* and *Gb* values. Therefore, different types of PSOs can be used for each particle in the main PSO. However, for simplicity and easier tracking, the same type of PSO is used for the particles of the main PSO in this study. The purpose of the main PSO is to achieve the best possible parameters of the ensemble classifier (i.e., BP), and concurrently utilise the inner PSOs to choose the best base-predictor for the ensemble and optimise the parameters of this chosen base-predictor.

Algorithm 2 Main PSO of ML-PSO

```
1. Create particles in the inner PSOs;
2. Initialise, with random values, the particles' positions, velocities, and personal best (Pb); and the swarm's global best
   (Gb) for the main PSO;
3. For each particle:
4.     Calculate the current fitness value by calling the inner PSO function;
5.     Update the particle's Pb and the Gb, if the current fitness value is better;
6. For each particle:
7.     Update their velocity and position using Eq. 1 and Eq. 2;
8. Update the inertia-weight, $w$, randomly;
9. If the particles' iterations exceed the maximum number of iteration or terminating criteria are met, stop and report Gb;
   else go to step 3;
```

Algorithm 3 Inner PSOs of ML-PSO

```
1. Create the particles. Initialise, with random values, their positions, velocities, and Pb; and the swarm's Gb;
2. For each particle:
3.     Calculate the current fitness value;
4.     Update Pb and Gb, if the current fitness value is better;
5. For each particle:
6.     Update their velocity and position using Eq. 1 and Eq. 2;
7. Update the inertia-weight, $w$, randomly;
8. If particles' iterations exceed the maximum number of iterations or terminating criteria are met, return Gb to the caller
   (Main PSO); else go to step 2;
```

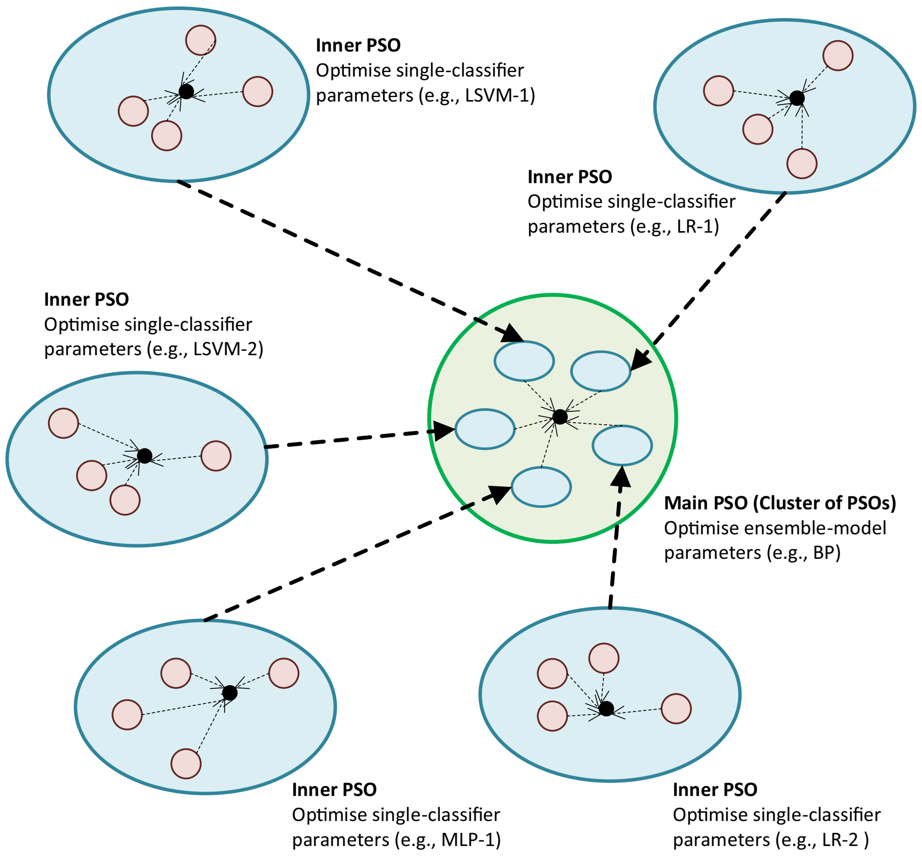
Inner PSO
Optimise single-classifier parameters (e.g., LSVM-1)
Inner PSO
Optimise single-classifier parameters (e.g., LR-1)
Inner PSO
Optimise single-classifier parameters (e.g., LSVM-2)
Main PSO (Cluster of PSOs)
Optimise ensemble-model parameters (e.g., BP)
Inner PSO
Optimise single-classifier parameters (e.g., MLP-1)
Inner PSO
Optimise single-classifier parameters (e.g., LR-2)

Fig. 1 Design of ML-PSO

After data preparation, the main PSO creates its PSO particles randomly, following which a classifier is assigned to each PSO particle to be processed as a base-predictor candidate. For instance, if the number of particles is 10 and given three classifiers to be investigated in our case—namely the LR, LSVM and MLP—these classifiers are distributed evenly among the PSO particles using a simple loop (4 PSOs for the LR, 3 PSOs for the LSVM, and 3 PSOs for the MLP). Each inner PSO runs independently without affecting the other inner PSOs. After an inner PSO process is terminated, it reports its *Gb* result to the main PSO, which uses this information to adjust the *Pb* of the particle and its own *Gb*. The particle initiates further inner PSO runs until the iterations are complete. The number of particles and iterations of main and inner PSOs can be set individually at the beginning of the process. The intention is to separate the optimisation process of ensemble parameters in the main PSO and parameter optimisation of the base-predictors of this ensemble in the inner PSOs. The reason for separation is that each predictor/classifier candidate has a different set of parameters. It is impossible to combine optimisation of the ensemble parameters and optimisation of the base-predictor candidate's parameters, since the number of parameters of each classifier is different (see the examples in Table 1).

Each inner PSO may optimise a different classifier, and each classifier has different sets of parameters. Initially, the classifier's parameters are set randomly and are used as the location and velocity vectors of the PSO. In each iteration, each particle calls the classifier assigned to it for evaluation using its current location vector as the classifier's parameters. Once the classifier completes its training and testing processes, it reports the measurement results (e.g., accuracy, precision, recall, F-measure) back to the PSO's particles for evaluation. Then, based on the evaluation results (the *Pb* and *Gb* vectors), each particle adjusts its velocity and location vectors. When all the particles have either completed their iterations or met their termination criteria, the inner PSOs send their classifier's identity, *Gb* vector and measurement values to their corresponding particle in the main PSO. The inner PSOs then use this information to adjust their velocity and location, and update the *Pb* and *Gb*. If the iterations are not completed (or termination criteria are not met), the particle will initiate another inner PSO and so on. Once the overall process is completed, the method reports the *Gb* set for the main PSO, which consists of its measurement value, ensemble-optimised parameters, the best base-predictor choice, and the optimised parameters of this base-predictor. See Algorithms 2 and 3 for details of the main PSO and inner PSOs, respectively.

Table 1 Parameters of classifiers to be optimised

Classifier	Parameter	Description	Type	Values
LSVM	C	Inverse of regularisation strength	Floating point	0.1–1.0
	Multi-class strategy	The multi-class strategy used if y contains more than two classes	Nominal	0 = ‘ovr’; 1 = ‘crammer singer’
	Intercept scaling	Intercept scaling is used to lessen the effect of regularisation on intercept (synthetic feature weight)	Floating point	0.1–1.0
	Max. Iteration	The maximum number of iterations to be run	Integer	100–3000
LR	C	The inverse of regularisation strength	Floating point	0.1–1.0
	Solver	Algorithm to use for optimisation	Nominal	0 = ‘newton-cg’; 1 = ‘lbfgs’; 2 = ‘liblinear’ 3 = ‘sag’
	Max. Iteration	The maximum number of iterations taken for the solvers to converge	Integer	50–300
MLP	Number of hidden layers	The number of hidden layers	Integer	1–3
	Number of hidden neurons	The number of hidden layer neurons; the number of neurons can be different for each hidden layer	Integer	50–3000
	Activation	Activation function for the hidden layer	Nominal	0 = ‘identity’ [f(x) = x] 1 = ‘logistic’ [f(x) = 1 / (1 + exp(-x))] 2 = ‘tanh’ [f(x) = tanh(x)] 3 = ‘relu’ [f(x) = max(0, x)]
	Solver	The solver for weight optimisation	Nominal	0 = ‘lbfgs’; 1 = ‘sgd’; 2 = ‘adam’
	Learning rate	The learning rate used when updating network weights	Floating Point	0.0001—0.05
	Shuffle	Whether to shuffle samples in each iteration	Boolean	0 = False; 1 = True
	Early stopping	Whether to use early stopping to terminate training when the validation score is not improving	Boolean	0 = False; 1 = True
	Beta 1	The exponential decay rate for estimates of the first-moment vector of Adam solver	Floating point	0.5–0.99999
	Beta 2	The exponential decay rate for estimates of the second-moment vector of Adam solver	Floating point	0.5–0.99999
BP	Base estimator	Predictor/classifier used to build the ensemble	Classifier	0 = LR; 1 = LSVM; 2 = MLP
	N estimators	The number of base estimators in the ensemble	Integer	1–30
	Max. samples	The number of samples to draw from the dataset to train each base estimator (n_samples = max_samples * total_samples)	Floating point	0.3–1.0
	Max. features	The number of features to draw from a set of features to train each base estimator (n_features = max_features * total_features)	Floating point	0.2–1.0
	Bootstrap	Whether samples are drawn with replacement	Boolean	0 = False; 1 = True
	Bootstrap features	Whether features are drawn with replacement	Boolean	0 = False; 1 = True

It should be noted that the ML-PSO in this study consists of two levels because it is used to optimise ensemble parameters, select the base-predictor for this ensemble, and optimise the parameters of the selected predictor. However, for more complex problems, and depending on the problem

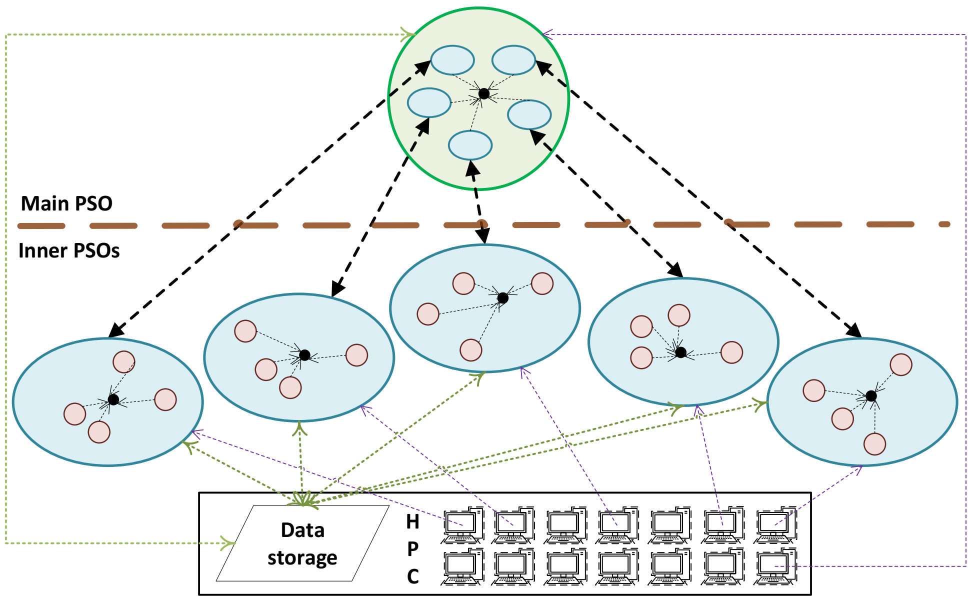

Fig. 2 Design of PML-PSO

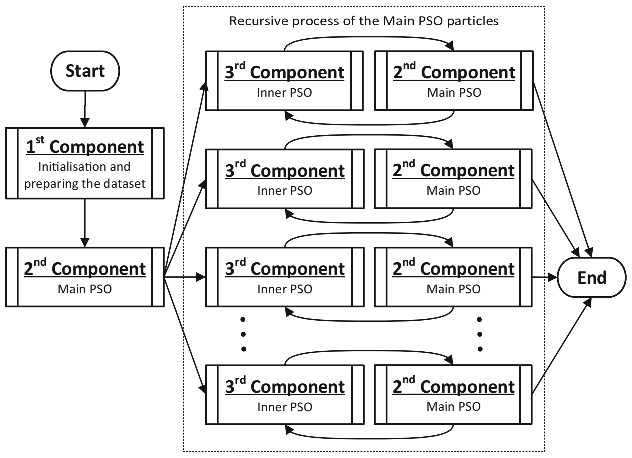
Recursive process of the Main PSO particles
Start
1st Component
Initialisation and preparing the dataset
2nd Component
Main PSO
3rd Component
Inner PSO
End

Fig. 3 PML-PSO processes

at hand, it can easily be expanded to comprise up to *n* levels.

3.4 Parallel multi-level PSO

The general problem with PSO and other nature-inspired optimisation algorithms is the large number of iterations required to simulate the 'evolution' process, resulting in long runtime. Approaches to overcome this problem include the simplification of the entire process, application of simpler equations, or parallelism [20, 28, 29, 38, 39, 42]. We utilise parallelism to design a parallel version of ML-PSO, PML-PSO in short, with the intention of speeding up the optimisation process. See Fig. 2 for a high-level representation of the method. The approach has been specifically designed to utilise grid computing facilities, such as the high-performance computing (HPC) facilities at the University of Newcastle (UoN), Australia. While PML-PSO has the same fundamental idea as ML-PSO, i.e., the utilisation of multiple levels of PSOs, it has been redesigned and rewritten to make the best use of parallelism in grid computing facilities. In any grid computing facility, each node is considered as a single CPU running independently of and in parallel with other nodes. The nodes do not share the RAM but the data storage. Each job/script submitted by a user is queued in a central job queue and is assigned to a CPU whenever one is free. Therefore, to fully utilise the power of HPC, PML-PSO is composed of several small programs that recursively call each other, as illustrated in Fig. 3.

Algorithm 4 First Component of PML-PSO

```
1.  Prepare all settings;
2.  Create the Settings and Lv1Gbest files (Lv1Gbest = -1) to save all settings and $Gb$ of main PSO, respectively;
3.  Create and queue a job for the second component and stop;
```

Algorithm 5 Second Component of PML-PSO

```
//The Second Component is used to handle the process of main PSO in PML-PSO
1.  Load the Settings and Lv1Gbest files;
2.  If this job is created by the First Component:
3.      Create n particles and set all their attributes as assigned in the settings;
4.      Create n Lv1ParticleClass files to save the n particles and all their attributes;
5.      Create n Lv1ParticleResult files with value -1;
6.      Create and queue n jobs for the Third Component and stop.
7.  Else if this job is created by the Third Component:
8.      particle = load the appropriate Lv1ParticleClass file;
9.      Lv1Gbest = load the Lv1Gbest file;
10.     If particle.iteration > particle.maxIteration or terminating criteria are met:
11.         Write Lv1Gbest to a report file then stop;
            //This is the end of this particle's cycle. Once all main PSO particles complete their cycles, the last report of
            Lv1Gbest in the report file is the expected solution
12.     Else:
13.         particle.currentFitness = load the Lv1ParticleResult file;
14.         Update particle.pbest with particle.currentFitness if particle.currentFitness > particle.pbest;
15.         Update Lv1Gbest with particle.currentFitness if particle.currentFitness > Lv1GBest;
16.         Update particle.velocity and particle.position using Eq. 1 and Eq. 2;
17.         Increment particle.iteration;
18.         Update the particleClass and Lv1Gbest files;
19.         Create and queue a Third Component job for this particular particle and stop.
            // The Second Component calls the Third Component for another round of inner PSOs. This instruction is similar
            with "Calculate" in the main PSO function of ML-PSO
```

Algorithm 6 Third Component of PML-PSO

```
//The Third Component is used to handle the process of inner PSOs in PML-PSO
1.  Load the appropriate Lv1ParticleResult and Lv1ParticleClass files for setting parameters of the job
2.  Create all particles. Then initialise, with random values, the particles' positions, velocities, and personal best (Pb), and
    the swarm's global best (Gb);
3.  For each particle:
4.      Calculate the current fitness value;
5.      Update Pb (particle) and Gb, if the current fitness value is better;
6.  For each particle:
7.      Update their velocities and positions using Eq. 1 and Eq. 2;
8.  Update the inertia-weight, w, randomly;
9.  If the particles' iterations exceed the maximum number of iterations or terminating criteria are met:
10.     If Gb is better than the Lv1ParticleResult file value, update the Lv1ParticleResult file;
11.     Create and queue a Second Component job corresponding to this particular particle and stop.
        //By creating the Second Component, this Third Component calls the main PSO to continue its iteration or stop when
        it completes the maximum number of iterations or when the terminating criteria are met.
```

PML-PSO consists of three different components, and each has been coded separately. The first component prepares the dataset, classifiers to be used in the process, and settings such as the number of particles and total iterations of both the main PSO and inner PSOs. Once the preparation phase has completed, this component creates and queues the job for the second component, which handles the main PSO. When the job for the second component is run on HPC, it creates and queues several jobs for the third component, corresponding to the particles of the main PSO. The jobs created for the third component represent the inner PSOs in ML-PSO. Each such job continuously updates its *Pb* and *Gb* values, adjusts the velocity and position of each particle, and so on until the number of scheduled iterations is completed or the termination criteria are met. Multithreading is utilised in the third component to speed up the entire process.

Once an inner PSO process has completed, its corresponding component (i.e., the third component) creates and queues a job for its parent (second) component using the available template. When this second component job is run, it updates *Pb* of the corresponding particle (i.e., the inner PSO), and *Gb* of the main PSO, and finally adjusts the velocity and position of the particle. Following this, if the number of iterations set for this particle (i.e., the inner PSO) has been completed or the termination criteria are met, the process for the particle is terminated; otherwise, another job for the next iteration of the inner PSO (i.e., the third component) is created and queued. The third component, thus created, will run as described earlier and terminates by creating a job for the parent (second)

Table 2 Accuracies of BP using default parameters versus ML-PSO with 5 particles and 10 iterations for each particle

Experiment	Default parameters								ML-PSO (5 particles)		
	GBo	AB	RF	BP (CART)[a]	BP (NNb)[a]	BP (LR)	BP (LSVM)	BP (MLP)	BP (LR)	BP (LSVM)	BP (MLP)
1	0.821	0.803	0.753	0.717	0.626	0.827	0.782	0.840	0.846	0.828	0.860
2	0.823	0.780	0.762	0.706	0.609	0.832	0.788	0.851	0.850	0.819	0.857
3	0.813	0.799	0.744	0.729	0.619	0.835	0.828	0.847	0.844	0.829	0.863
4	0.818	0.782	0.756	0.724	0.634	0.839	0.781	0.837	0.848	0.825	0.855
5	0.825	0.800	0.764	0.726	0.606	0.834	0.795	0.852	0.845	0.833	0.869
6	0.823	0.789	0.744	0.729	0.628	0.820	0.788	0.841	0.848	0.814	0.864
7	0.826	0.807	0.750	0.694	0.651	0.825	0.799	0.834	0.849	0.819	0.868
8	0.818	0.806	0.743	0.723	0.631	0.844	0.804	0.838	0.850	0.832	0.858
9	0.813	0.799	0.758	0.739	0.634	0.836	0.803	0.843	0.850	0.828	0.857
10	0.819	0.800	0.751	0.736	0.631	0.831	0.786	0.844	0.849	0.822	0.862
Avg:	0.820	0.797	0.753	0.722	0.627	0.832	0.795	0.843	0.848	0.825	0.861
Std.Dev	0.004	0.009	0.007	0.013	0.012	0.007	0.013	0.006	0.002	0.006	0.005

[a]BP uses CART and NNb algorithms as its default base-predictors [31, 77]

component. Thus, the second and third components are designed to call each other in a double recursive fashion.

The communication between the components, such as passing parameters and global variables for *Pb* and *Gb*, is done by saving them on several temporary files in a common data storage on the HPC. Theoretically, all particles in the main PSO will run in parallel without waiting for the other particles to complete. Therefore, in an ideal situation, where the HPC's CPUs immediately serve all jobs at the same time, the processing time of the main PSO is equal to the longest processing time of its particles; thus solving the problem of long runtime. See Algorithms 4, 5 and 6 for details of the first, second and third components, respectively.

4 Experiments and results

As discussed earlier, in our experiments, we used the LR, LSVM and MLP as base-predictors of BP. In addition, we also compared the results with those of CART and NNb algorithms, which are the default base-predictors for BP [31]. It is worth noting that other types of ensemble models, such as Gradient Boosting (GBo) [74] and Random Forest (RF) [70], are limited to using CART as their base-predictor. They were, therefore, not considered suitable for our current study. Another popular ensemble model, Adaptive Boosting (AB) [75], which has more than one suitable base-predictor, cannot use the MLP as its base-predictor [4]; we also found its performance for sentiment analysis to be inferior to BP [4].

The parameters to be optimised for each classifier can be found in Table 1. In order to facilitate comparison with results of previous experiments, fixed settings for the inner PSOs were used. These settings are: 15 particles and 30 iterations for each particle; c1 = c2 = 1.49445; and a random number for *w* (0.5–0.9) generated every time the velocity was updated. Regarding the dataset, 1000 records from the Yelp 2017 review dataset [76] and 1000 features of two polarities (negative/positive) were used. Since the Yelp review dataset is an unlabelled dataset, we used star ratings given by its reviewers as the basis for assigning negative (1 & 2 stars) and positive (3, 4 and 5 stars) polarities. For detailed information about this decision, please refer to [4]. We used 10-fold cross validation for training and testing every classifier in the inner PSOs (particles of the main PSO).

4.1 Experiments on ML-PSO

The first series of experiments was conducted by varying the number of particles in the main PSO from 5 to 50, and each particle was run for 10 iterations. Fitness of each particle was measured using the accuracy achieved with the parameter settings of the classifiers. The maximum runtime of each job was limited to 400 h (for the UoN's HPC facility), and the maximum RAM available to each job was 120 Gigabytes. Detailed results of each process were recorded, including the *Gb* of every particle in the main PSO, with each particle being an inner PSO.

Results in Table 2 show that parameters determined by ML-PSO, using only 5 particles for the main PSO, outperform the default parameters by a minimum of 1.6% and

Table 3 Average difference and MW U test results

ML-PSO (5p) optimised params	Default parameters	Average difference (%)	MW U test p value ($\alpha < 0.01$)
BP(LR)	BP(LR)	1.6	0.00022
BP(LR)	GBo	2.8	0.00018
BP(LR)	AB	5.1	0.00018
BP(LR)	RF	9.5	0.00018
BP(LSVM)	BP(LSVM)	3	0.001
BP(LSVM)	GBo	0.5	0.05876
BP(LSVM)	AB	2.8	0.00018
BP(LSVM)	RF	7.2	0.00018
BP(MLP)	BP(MLP)	1.8	0.00018
BP(MLP)	BP(LR)	2.9	0.00018
BP(MLP)	BP(LSVM)	6.6	0.00018
BP(MLP)	BP(CART)	13.9	0.00018
BP(MLP)	BP(NNb)	23.4	0.00018
BP(MLP)	GBo	4.1	0.00018
BP(MLP)	AB	6.4	0.00018
BP(MLP)	RF	10.8	0.00018

up to 23.4%. Accuracies of the optimised BP(LR), BP(MLP) and BP(LSVM) are higher by 1.6%, 1.8% and 3%, respectively, compared to their counterparts with default parameters [77]. The optimised BP(LR), BP(MLP) and BP(LSVM) also outperform other ensemble models, namely GBo, AB and RF with default parameters, by at least 2.8%, 4.1% and 0.5%, respectively. The MLP classifier, which produces the best results overall, outperforms the default base-predictors, Decision Tree/CART and NNb algorithms, by 13.9% and 23.4%, respectively. These results indicate that the proposed method, ML-PSO, can greatly improve the performance of the sentiment polarity predictors, even with a small number of particles. Statistical analysis using the Mann–Whitney (MW) U test, as shown in Table 3, confirmed that the improvements are mostly significant, since all of the p-values of the tests are below the significance level ($\alpha < 0.01$), except the optimised BP(LSVM) against GBo.

To further investigate improvements that might be achieved with more particles in the main PSO, the above experiments were repeated by increasing the number of particles up to 50, with an interval of 5 particles for each experiment. The results show that, with more particles, ML-PSO is able to further improve the accuracy of the prediction by $\sim$ 1.2%, from a maximum of 0.869 to 0.881 (see Table 4, the *Gb* of ML-PSO (accuracy), sub-column BP(MLP)). Another interesting observation made from these results is that the MLP is always the best base-predictor for BP, except for the ML-PSO with 25 particles, where BP(LR) is slightly better than BP(MLP). From our results, it can, therefore, be concluded that the proposed ML-PSO method can choose a better base-predictor automatically for the ensemble algorithm (BP), and successfully optimise the parameters of the chosen base-predictor as well as the parameters of the ensemble.

However, upon closer inspection of the processing times of ML-PSO, it was discovered that the processing times for 10 particles and above are the same, i.e., 400 h (see Table 4, the “Processing time” column, ML-PSO). 400 h, as mentioned earlier, is the maximum processing time available to each node in the HPC facility at the UoN. A detailed inspection of the process logs revealed that most of the particles in the main PSO neither completed all iterations nor met the termination criteria; rather, they were forcefully terminated after reaching the maximum processing time limit. It should be noted that, even though the processes were forcefully terminated, ML-PSO still provides an output because the current *Gb* and the vectors that yielded it are recorded at each iteration. Nevertheless, it is safe to assume that the results are not the best possible results as initially expected. To overcome this situation, experiments using parallelism (i.e., PML-PSO) were conducted, and the results are described in the next section.

4.2 Experiments on PML-PSO

PML-PSO has been created, as explained in earlier sections, to overcome the problem faced by ML-PSO in the preceding section. Experiments to test this idea were run on the same HPC facility at the UoN, which has 2560 cores for 66 CPU and 4 GPU nodes, and up to 512 Gigabytes' RAM available to be assigned to each node. For all our

Table 4 ML-PSO versus PML-PSO, in a non-ideal environment

Number of particles in main PSO	Processing time (hour)		*Gb* of ML-PSO (accuracy)			*Gb* of PML-PSO (accuracy)		
	ML-PSO	PML-PSO	BP (LR)	BP (LSVM)	BP (MLP)	BP (LR)	BP (LSVM)	BP (MLP)
50	400.00	878.59	0.870	0.859	0.873	0.874	0.874	0.884
45	400.00	705.24	0.863	0.853	0.876	0.871	0.870	0.886
40	400.00	622.36	0.861	0.868	0.878	0.873	0.871	0.883
35	400.00	134.31	0.860	0.865	0.873	0.872	0.873	0.879
30	400.00	119.92	0.862	0.864	0.881	0.876	0.876	0.885
25	400.00	807.82	0.870	0.861	0.868	0.872	0.872	0.881
20	400.00	126.95	0.865	0.856	0.875	0.872	0.871	0.881
15	400.00	557.76	0.860	0.868	0.872	0.871	0.873	0.879
10	400.00	512.50	0.863	0.857	0.875	0.869	0.869	0.880
5	245.77	123.22	0.845	0.833	0.869	0.871	0.861	0.875
MW U test *p* value (ML-PSO to PML-PSO, $\alpha < 0.01$):						0.00034	0.00068	0.00168

Table 5 ML-PSO versus PML-PSO without the MLP, in an ideal environment

Number of particles in main PSO	Processing time (hour)		Highest *Gb* and the accuracy			
	ML-PSO	PML-PSO	ML-PSO *Gb*	ML-PSO accuracy	PML-PSO *Gb*	PML-PSO accuracy
50	125.39	46.59	BP(LR)	0.872	BP(LR)	0.870
45	81.86	40.92	BP(LSVM)	0.871	BP(LR)	0.870
40	94.82	49.27	BP(LR)	0.872	BP(LR)	0.870
35	141.26	46.73	BP(LR)	0.870	BP(LSVM)	0.871
30	128.47	42.45	BP(LR)	0.869	BP(LR)	0.869
25	108.36	45.38	BP(LR)	0.867	BP(LR)	0.870
20	39.33	44.63	BP(LR)	0.871	BP(LR)	0.872
15	34.37	44.02	BP(LR)	0.865	BP(LR)	0.870
10	8.65	47.32	BP(LR)	0.868	BP(LSVM)	0.869
5	3.83	39.80	BP(LR)	0.861	BP(LR)	0.862

MW U test analysis for the accuracy between ML-PSO and PML-PSO ($\alpha < 0.01$), *p* value = 0.8181

experiments, we used an HPC cluster consisting of 32 nodes of CPUs. Each experimental setting was run only once, for two different HPC load environments, since each optimisation process took a very long time to complete. Table 4 presents the results of experiments run in a non-ideal environment, where HPC loads range from normal to full and jobs have to wait in the job queue before executing. Similarly, Table 5 presents the results for similar experiments, but without BP(MLP), repeated in an ideal environment where the HPC load was low enough for all jobs to be executed immediately. It is also worth mentioning here that, since a node's failure can prevent the successful completion of a job, node failure can impact the prediction accuracy. However, node failure is extremely rare in the HPC environment, and our job logs did not report any node failures during our experiments.

Results, as can be seen in Table 4 under the "Processing time" column, show that PML-PSO is able to successfully overcome the processing time limitation of 400 h (note that the processing time values for PML-PSO are the total hours required by all the jobs). One interesting observation is that, for particle sizes 20, 30 and 35, the processing times are similar to those with only 5 particles. The reason for this is that, in these cases, the HPC job queue was sufficiently low enough to allow the PML-PSO processes to run in a truly parallel manner. In the other cases, however, the HPC load prevented truly parallel processing of all PSO runs. For further understanding of how the PML-PSO

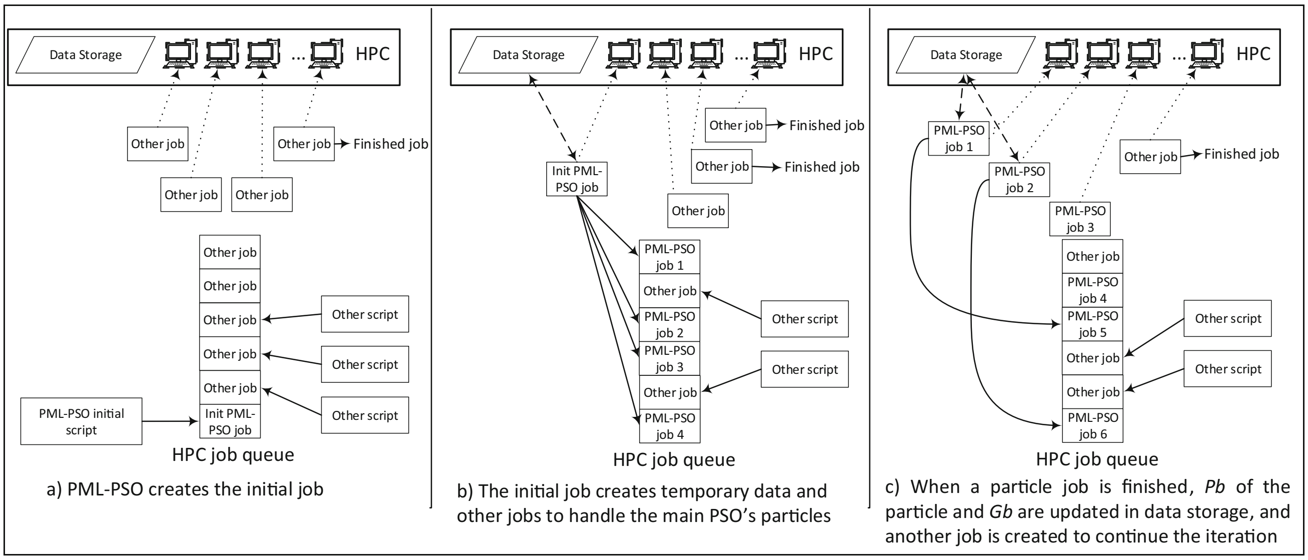
Data Storage
HPC
Other job
Finished job
Other script
PML-PSO initial script
Init PML-PSO job
HPC job queue
a) PML-PSO creates the initial job
PML-PSO job 1
PML-PSO job 2
PML-PSO job 3
PML-PSO job 4
PML-PSO job 5
PML-PSO job 6
b) The initial job creates temporary data and other jobs to handle the main PSO's particles
c) When a particle job is finished, *Pb* of the particle and *Gb* are updated in data storage, and another job is created to continue the iteration

Fig. 4 The PML-PSO process in detail

works in HPC together with other jobs, please refer to the illustration in Fig. 4.

From the results of the MW U test listed in Table 4, it can be seen that accuracy improvements with PML-PSO are significantly higher than those with ML-PSO. This indicates that the processing time limitation placed on HPC has a severe impact on the efficacy of ML-PSO and its goal of achieving the best possible solution. The results also validated the idea of adding parallelism to ML-PSO, since PML-PSO was able to meet its goal of speeding up the process, evading HPC processing time limitations and performing multi-level optimisation like ML-PSO.

To conduct a fairer comparison of ML-PSO and PML-PSO, we conducted another set of experiments by excluding BP(MLP), which requires much longer time to reach convergence than other classifiers. The assumption was that ML-PSO would successfully terminate under 400 h without BP(MLP). Besides the exclusion of BP(MLP), other settings were the same as previous experiments. Efforts were also made to run the experiments when the HPC load was low enough to allow all jobs from PML-PSO to be served immediately. The results of these experiments can be found in Table 5.

These results in Table 5 show that the best accuracies provided by ML-PSO and PML-PSO are very similar, which means that, without being cut off by the processing time limitation of HPC, ML-PSO can match the performance of PML-PSO. MW U test analysis between the accuracies of ML-PSO and PML-PSO confirmed that their difference is not significant, since the p-value is higher than the significance level of $\alpha < 0.01$. Nevertheless, since it uses iterations instead of parallelism, ML-PSO needed more time to reach its stopping condition when we increased the number of particles; whereas the processing times are similar for all particle sizes in PML-PSO. In an ideal environment, parallelism allows all or almost all the particles of PML-PSO entering the job queue to run at the same time. The processing times of PML-PSO experiments were, however, longer than ML-PSO when the number of particles is relatively small (5 to 20). This is because, every time the component is run, it should upload the sample features from data storage to memory, whereas in ML-PSO, the sample features are processed only once and stay in memory for the duration of training.

The processing times for ML-PSO with 40, 45 and 50 particles are inconsistent with the trend of increased processing time for a higher number of particles. An educated guess here is that this is caused by the variance of convergence speed of classifiers used inside the inner PSO; however, we do not have a technical explanation, since we did not record the time required by the classifiers of the inner PSO particles. Nonetheless, from the results in Table 5, we can conclude that the processing times for PML-PSO are almost always constant in an ideal situation, whereas the processing time increases with an increase in the number of particles in ML-PSO.

4.3 Experiments using large-scale data

Our last set of experiments was conducted by applying the best results of PML-PSO on 500,000 records from the Yelp review dataset. The experiment settings and pre-processing steps were based on previous research [4], and each experimental setting was repeated 10 times. The experiments were run using 10-fold cross validation. Experiment types were (A) 1 and 2 stars as negative polarity and 3, 4 and 5 as positive polarity; (B) 1, 2 and 3 stars as negative polarity and 4 and 5 as positive polarity; and (C) 1 and 2 stars as negative polarity, 3 star as neutral polarity, and 4 and 5 as positive polarity.

As can be seen from the results in Table 6, PML-PSO can indeed improve the accuracy and other measurements such as precision, recall and F1 (F-measure) of BP to predict sentiment polarity of customer reviews. While training the classifier with a larger dataset can improve its performance, using our methods, we can be sure of achieving the best possible parameters and also the best base-classifier to be used for the ensemble. Based on the results in Table 6, the improvement achieved by applying PML-PSO (see BP(MLP)) is quite high compared to the results obtained using default parameters and CART as the default base-predictor (see BP(CART)). The improvements on all measurements are around 6%, 7.7% and 7.5% for Type A, B and C experiments, respectively. Statistical analysis using the MW U test with a significance level of $\alpha < 0.01$ confirms that the accuracy improvements achieved from PML-PSO are significant when compared with BP(LSVM), BP(LR) and BP(MLP) with default settings. The best improvement is achieved by BP(LSVM): the performance of BP(LSVM) is similar to BP(LR) using PML-PSO optimised parameters, whereas its results are worse than BP(LR) with default parameters.

5 Conclusion

Ensemble models, such as BP, can provide better performance than a single classifier. However, determining a suitable classifier for the ensemble's base-predictor is a challenging problem. Other difficulties for problems such as sentiment polarity detection include obtaining optimal parameters for both the ensemble and its base-predictor. The methods proposed in this paper, ML-PSO and PML-PSO, can overcome all of the above-mentioned problems. The first set of experiments, using a small number of particles in the main PSO and a small number of records (1000 records), showed that the proposed ML-PSO can outperform the default base-classifier settings for BP (CART or NNb). The accuracy of ML-PSO optimised BP ensemble is 14% and 23% higher than BP(CART) and BP(NNb), respectively. Our experimental results also showed that the accuracy could be further improved by using more particles in ML-PSO.

However, it was observed from the experiments that the limitation on processing times on HPC prevents ML-PSO processes from running until completion and providing better results. The second method proposed in this paper, PML-PSO, utilises parallelism to reduce the processing time of ML-PSO and helps overcome the problem caused by limited processing times available on HPC. By breaking down a long process into several smaller and independent processes running in parallel, PML-PSO requires less time for successful completion even with a larger number of particles. Therefore, PML-PSO can successfully obtain

Table 6 Performance of PML-PSO optimised parameters for processing big data

Experiment type	Classifier	Default parameters					PML-PSO optimised parameters					MW U test (p value)[b]	
		Acc	StD[a]	Prec	Rec	F1	Acc	StD[a]	Prec	Rec	F1	Acc	F1
A (2-polarity)	BP (CART)	0.864	0.0066	0.864	0.864	0.864	–	–	–	–	–		
	BP (LSVM)	0.898	0.0001	0.897	0.898	0.897	0.916	0.0009	0.913	0.916	0.913	0.00018	0.00018
	BP (LR)	0.909	0.0001	0.908	0.909	0.908	0.917	0.0009	0.915	0.917	0.915	0.00018	0.00018
	BP (MLP)	0.917	0.0004	0.916	0.917	0.917	0.923	0.0003	0.921	0.923	0.921	0.00018	0.00018
B (2-polarity)	BP (CART)	0.814	0.0068	0.817	0.814	0.815	–	–	–	–	–		
	BP (LSVM)	0.854	0.0001	0.853	0.854	0.853	0.885	0.0014	0.884	0.885	0.883	0.00018	0.00018
	BP (LR)	0.873	0.0001	0.872	0.873	0.872	0.886	0.0014	0.885	0.886	0.884	0.00018	0.00018
	BP (MLP)	0.883	0.0004	0.882	0.883	0.882	0.891	0.0002	0.890	0.891	0.890	0.00018	0.00018
C (3-polarity)	BP (CART)	0.775	0.0064	0.756	0.775	0.761	–	–	–	–	–		
	BP (LSVM)	0.819	0.0001	0.798	0.819	0.805	0.836	0.0019	0.811	0.836	0.809	0.00018	0.00018
	BP (LR)	0.829	0.0002	0.809	0.829	0.815	0.839	0.0013	0.815	0.839	0.817	0.00018	0.00252
	BP (MLP)	0.840	0.0004	0.827	0.840	0.832	0.850	0.0001	0.833	0.850	0.835	0.00018	0.00018

[a]StD = standard deviation of the accuracy of experiments over 10 runs

[b]We did not include BP(CART) in the MW U test, since we did not investigate its optimised version; the test was applied to the detailed results of 10-fold cross validation of each experiment

better solutions than those obtained by ML-PSO even under the restrictions placed on the available processing time. However, further investigation revealed that, when both methods were applied in an ideal environment, their performances were very similar; thus, proving that both methods are quite similar except in terms of their implementation and how they deal with a larger number of particles. The final set of experiments using a larger dataset (500,000 records) proved that the best possible parameter settings obtained using PML-PSO can significantly improve the accuracy of sentiment polarity prediction.

For future work, we plan to implement ML-PSO and PML-PSO for optimising the classifiers to detect fake reviews or other cases.

Acknowledgements The first author would like to acknowledge financial support from the Indonesian Endowment Fund for Education (LPDP), Ministry of Finance; and the Directorate General of Higher Education (DIKTI), Ministry of Education and Culture, Republic of Indonesia.

References

1. Li, X., Li, J., Wu, Y.: A global optimization approach to multi-polarity sentiment analysis. PLoS ONE **10**(4), e0124672 (2015). https://doi.org/10.1371/journal.pone.0124672
2. Huang, J., Xue, Y., Hu, X., Jin, H., Lu, X., Liu, Z.: Sentiment analysis of Chinese online reviews using ensemble learning framework. Clust. Comput. **22**, 3043–3058 (2019)
3. Giatsoglou, M., Vozalis, M.G., Diamantaras, K., Vakali, A., Sarigiannidis, G., Chatzisavvas, K.C.: Sentiment analysis leveraging emotions and word embeddings. Expert Syst. Appl. **69**, 214–224 (2017). https://doi.org/10.1016/j.eswa.2016.10.043
4. Budhi, G.S., Chiong, R., Pranata, I., Hu, Z.: Predicting rating polarity through automatic classification of review texts. In: Proceedings of the 2017 IEEE Conference on Big Data and Analytics (ICBDA), pp. 19-24. Kuching, Malaysia, 16–17 November (2017)
5. Chiong, R., Fan, Z., Hu, Z., Adam, M.T.P., Lutz, B., Neumann, D.: A sentiment analysis-based machine learning approach for financial market prediction via news disclosures. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO '18 Companion), pp. 278–279. Kyoto, Japan, 15–19 July (2018)
6. Bansal, P., Kaur, R.: Twitter sentiment analysis using machine learning and optimization techniques. Int. J. Comput. Appl. **179**(19), 5–8 (2018)
7. Yousefpour, A., Ibrahim, R., Hamed, H.N.A.: Ordinal-based and frequency-based integration of feature selection methods for sentiment analysis. Expert Syst. Appl. **75**, 80–93 (2017). https://doi.org/10.1016/j.eswa.2017.01.009
8. Zhang, W., Kong, S.-X., Zhu, Y.-C., Wang, X.: Sentiment classification and computing for online reviews by a hybrid SVM and LSA based approach. Clust. Comput. **22**, 1–14 (2018). https://doi.org/10.1007/s10586-017-1693-7
9. Bagheri, A., Saraee, M., de Jong, F.: Care more about customers: Unsupervised domain-independent aspect detection for sentiment analysis of customer reviews. Knowl.-Based Syst. **52**, 201–213 (2013). https://doi.org/10.1016/j.knosys.2013.08.011
10. Fersini, E., Messina, E., Pozzi, F.A.: Expressive signals in social media languages to improve polarity detection. Inf. Process. Manag. **52**(1), 20–35 (2016). https://doi.org/10.1016/j.ipm.2015.04.004
11. Devika, M.D., Sunitha, C., Ganesh, A.: Sentiment analysis: A comparative study on different approaches. Procedia Comput. Sci. **87**, 44–49 (2016). https://doi.org/10.1016/j.procs.2016.05.124
12. Sonagi, A., Gore, D.: Efficient sentiment analysis using hybrid PSO-GA approach. Int. J. Innov. Res. Comput. Commun. Eng. **5**(6), 11910–11916 (2017). https://doi.org/10.15680/IJIRCCE.2017
13. Shang, L., Zhou, Z., Liu, X.: Particle swarm optimization-based feature selection in sentiment classification. Soft Comput. **20**(10), 3821–3834 (2016). https://doi.org/10.1007/s00500-016-2093-2
14. Basari, A.S.H., Hussin, B., Ananta, I.G.P., Zeniarja, J.: Opinion mining of movie review using hybrid method of support vector machine and particle swarm optimization. Procedia Eng. **53**, 453–462 (2013). https://doi.org/10.1016/j.proeng.2013.02.059
15. Cho, M.Y., Hoang, T.T.: Feature selection and parameters optimization of SVM using particle swarm optimization for fault classification in power distribution systems. Comput. Intell. Neurosci. **2017**, 1–9 (2017). https://doi.org/10.1155/2017/4135465
16. Weise, T., Zapf, M., Chiong, R., Nebro, A.J.: Why is optimization difficult? In: Chiong, R. (ed.) Nature-Inspired Algorithms for Optimisation, pp. 1–50. Springer, Berlin (2009)
17. Zolghadr-Asli, B., Bozorg-Haddad, O., Chu, X.: Introduction. In: Bozorg-Haddad, O. (ed.) Advanced Optimization by Nature-Inspired Algorithms. Springer, Singapore (2018)
18. Moser, I., Chiong, R.: Dynamic function optimization: The moving peaks benchmark. In: Alba, E., Nakib, A., Siarry, P. (eds.) Metaheuristics for Dynamic Optimization, pp. 35–59. Springer, Berlin (2013)
19. Lung, R.I., Dumitrescu, D.: Evolutionary swarm cooperative optimization in dynamic environments. Nat. Comput. **9**(1), 83–94 (2010). https://doi.org/10.1007/s11047-009-9129-9
20. Khoshahval, F., Zolfaghari, A., Minuchehr, H., Abbasi, M.R.: A new hybrid method for multi-objective fuel management optimization using parallel PSO-SA. Prog. Nucl. Energy **76**, 112–121 (2014). https://doi.org/10.1016/j.pnucene.2014.05.014
21. Abedi, M., Chiong, R., Noman, N., Zhang, R.: A hybrid particle swarm optimisation approach for energy-efficient single machine scheduling with cumulative deterioration and multiple maintenances. In: Proceedings of 2017 IEEE Symposium Series on Computational Intelligence (SSCI), pp. 2930–2937. Honolulu, Hawaii, USA, 27 November–1 December (2017)
22. Hu, Z., Chiong, R., Pranata, I., Susilo, W., Bao, Y.: Identifying malicious web domains using machine learning techniques with online credibility and performance data. In: Proceedings of IEEE Congress on Evolutionary Computation (CEC), pp. 5186–5194. Vancouver, BC, Canada, 24-29 July (2016)
23. Hu, Z., Chiong, R., Pranata, I., Bao, Y., Lin, Y.: Malicious web domain identification using online credibility and performance data by considering the class imbalance issue. Ind. Manag. Data Syst. **119**(3), 676–696 (2019). https://doi.org/10.1108/IMDS-1102-2018-0072
24. Cao, J., Cui, H., Shi, H., Jiao, L.: Big data: A parallel particle swarm optimization back propagation neural network algorithm based on MapReduce. PLoS ONE **11**(6), 1–17 (2016). https://doi.org/10.1371/journal.pone.0157551
25. Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., Lin, C.-J.: LIBLINEAR: A library for large linear classification. J. Mach. Learn. Res. **9**, 1871–1874 (2008)
26. Budhi, G.S., Chiong, R., Hu, Z., Pranata, I., Dhakal, S.: Multi-PSO based classifier selection and parameter optimisation for

sentiment polarity prediction. In: Proceedings of IEEE Conference on Big Data and Analytics (ICBDA), pp. 68–73. Langkawi Island, Malaysia, 21-22 November (2018)

27. Souza, E., Santos, D., Oliveira, G., Silva, A., Oliveira, A.L.I.: Swarm optimization clustering methods for opinion mining. Nat. Comput. (2018). https://doi.org/10.1007/s11047-018-9681-2
28. Wu, K., Zhu, Y., Li, Q., Han, G.: Algorithm and implementation of distributed ESN using spark framework and parallel PSO. Appl. Sci. **7**(4), 353 (2017). https://doi.org/10.3390/app7040353
29. Szwed, P., Chmiel, W.: Multi-swarm PSO algorithm for the quadratic assignment problem: a massive parallel implementation on the OpenCL platform. In: arXiv:1504.05158. (2015)
30. Lalwani, S., Sharma, H., Satapathy, S.C., Deep, K., Bansal, J.C.: A survey on parallel particle swarm optimization algorithms. Arab. J. Sci. Eng. **44**(4), 2899–2923 (2019). https://doi.org/10.1007/s13369-018-03713-6
31. Breiman, L.: Bagging predictors. Mach. Learn. **24**(2), 123–140 (1996). https://doi.org/10.1007/bf00058655
32. Menard, S.: Logistic Regression: From Introductory to Advanced Concepts and Applications. SAGE, Los Angeles (2010)
33. Campbell, C., Ying, Y.: Learning with Support Vector Machines. Morgan & Claypool, San Rafael (2011)
34. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning internal representations by error propagation. Parallel Distributed Processing: Explorations in the Microstructure of Cognition vol 1, pp. 318–362. MIT Press, Cambridge (1986)
35. Onan, A., Korukoğlu, S., Bulut, H.: A multiobjective weighted voting ensemble classifier based on differential evolution algorithm for text sentiment classification. Expert Syst. Appl. **62**, 1–16 (2016). https://doi.org/10.1016/j.eswa.2016.06.005
36. Kennedy, J., Eberhart, R.C.: Particle Swarm Optimization. In: Proceedings of the IEEE International Conference on Neural Networks, pp. 1942–1948. Perth, Australia, 27 November–1 December (1995)
37. Kennedy, J., Eberhart, R.C.: A discrete binary version of the particle swarm algorithm. In: Proceedings of the IEEE International Conference on Systems, Man & Cybernetics Computational Cybernetics & Simulation, (5), pp. 4104–4108. Orlando, FL, USA, 12–15 October (1997)
38. Yang, X.S., Deb, S., Fong, S.: Accelerated Particle Swarm Optimization and Support Vector Machine for business optimization and applications. Networked Digital Technologies (NDT2011). Commun. Comput. Inf. Sci. **136**, 53–66 (2011)
39. Tan, Y., Zhang, J.: Magnifier particle swarm optimization. In: Chiong, R. (ed.) Nature-Inspired Algorithms for Optimisation, pp. 279–298. Springer, Berlin (2009)
40. Lynn, N., Suganthan, P.N.: Ensemble particle swarm optimizer. Appl. Soft Comput. **55**, 533–548 (2017). https://doi.org/10.1016/j.asoc.2017.02.007
41. Wahyudi, M., Kristiyanti, D.A.: Sentiment analysis of smartphone product review using support vector machine algorithm-based particle swarm optimization. J. Theor. Appl. Inf. Technol. **91**(1), 189–201 (2016)
42. Kumar, S., Kumar, H.: Rapid PSO based features selection for classification. Int. J. Adv. Res. Comput. Sci. **8**(9), 682–690 (2017). https://doi.org/10.26483/ijarcs.v8i9.5173
43. Nazir, M., Majid-Mirza, A., Ali-Khan, S.: PSO-GA based optimized feature selection using facial and clothing information for gender classification. J. Appl. Res. Technol. **12**(1), 145–152 (2014). https://doi.org/10.1016/S1665-6423(14)71614-1
44. Mikula, M., Machová, K.: Combined approach for sentiment analysis in Slovak using a dictionary annotated by particle swarm optimization. Acta Electrotech. Inf. **18**(2), 27–34 (2018). https://doi.org/10.15546/aeei-2018-0013
45. Gaspar, R., Pedro, C., Panagiotopoulos, P., Seibt, B.: Beyond positive or negative: Qualitative sentiment analysis of social media reactions to unexpected stressful events. Comput. Hum. Behav. **56**, 179–191 (2016). https://doi.org/10.1016/j.chb.2015.11.040
46. Nelder, J.A., Wedderburn, R.W.M.: Generalized linear models. J. R. Stat. Soc. **135**(3), 370–384 (1972). https://doi.org/10.2307/2344614
47. Hastie, T., Tibshirani, R.: Generalized Additive Models. Chapman and Hall/CRC, Boca Raton (1990)
48. Dunteman, G.H., Ho, M.H.R.: Generalized linear models. In: Dobson, A.J., Barnett, A.G. (eds.) An Introduction to Generalized Linear Models, pp. 2–6. SAGE Publications, Thousand Oaks (2011)
49. Dobson, A.J., Barnett, A.G.: An Introduction to Generalized Linear Models, 3rd edn. CRC Press, Boca Raton (2008)
50. Yu, D., Mu, Y., Jin, Y.: Rating prediction using review texts with underlying sentiments. Inf. Process. Lett. **117**, 10–18 (2017). https://doi.org/10.1016/j.ipl.2016.08.002
51. Shah, Y.S., Hernandez-Garcia, L., Jahanian, H., Peltier, S.J.: Support vector machine classification of arterial volume-weighted arterial spin tagging images. Brain Behav. **6**, 1–8 (2016)
52. Sun, J., Fujita, H., Chen, P., Li, H.: Dynamic financial distress prediction with concept drift based on time weighting combined with Adaboost support vector machine ensemble. Knowl.-Based Syst. **120**, 4–14 (2017)
53. Chinniyan, K., Gangadharan, S., Sabanaikam, K.: Semantic similarity based web document classification using support vector machine. Int. Arab J. Inf. Technol. **14**(3), 285–293 (2017)
54. Lo, S.L., Chiong, R., Cornforth, D.: Using support vector machine ensembles for target audience classification on Twitter. PLoS ONE **10**(4), e0122855 (2015)
55. Lo, S.L., Cornforth, D., Chiong, R.: Identifying the high-value social audience from Twitter through text-mining methods. In: Proceedings of the 18th Asia Pacific Symposium on Intelligent and Evolutionary Systems, pp. 325–339. Singapore, 10–12 November (2014)
56. Hur, M., Kang, P., Cho, S.: Box-office forecasting based on sentiments of movie reviews and independent subspace method. Inf. Sci. **372**, 608–624 (2016). https://doi.org/10.1016/j.ins.2016.08.027
57. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the 13th International Conferenceon Artificial Intelligence and Statistics, pp. 249–256. Sardinia, Italy, 13–15 May (2010)
58. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. CoRR abs/1412.6980 (2014)
59. Adipranata, R., Budhi, G.S., Setiahadi, B.: Automatic classification of sunspot groups for space weather analysis. Int. J. Multimed. Ubiquitous Eng. **8**(3), 41–54 (2013)
60. Budhi, G.S., Adipranata, R.: Handwritten Javanese character recognition using several artificial neural network methods. J. ICT Res. Appl. **8**(3), 195–212 (2015). https://doi.org/10.5614/itbj.ict.res.appl.2015.8.3.2
61. Budhi, G.S., Adipranata, R.: Java characters recognition using evolutionary neural network and combination of Chi2 and backpropagation neural network. Int. J. Appl. Eng. Res. **9**(22), 18025–18036 (2014)
62. Sangjae, L., Joon, Y.C.: Predicting the helpfulness of online reviews using multilayer perceptron neural networks. Expert Syst. Appl. **41**(6), 3041–3046 (2014)
63. Quinlan, J.R.: Induction of decision trees. Mach. Learn. **1**(1), 81–106 (1986). https://doi.org/10.1007/bf00116251
64. Hunt, E.B., Marin, J., Stone, P.J.: Experiments in Induction. Academic Press, New York (1966)
65. Rokach, L., Maimon, O.: Data Mining with Decision Trees: Theory and Applications. World Scientific Publishing Company, Singapore (2007)

66. Luo, B., Zeng, J., Duan, J.: Emotion space model for classifying opinions in stock message board. Expert Syst. Appl. **44**, 138–146 (2016). https://doi.org/10.1016/j.eswa.2015.08.023
67. Xu, Z., Li, P., Wang, Y.: Text classifier based on an improved SVM decision tree. Phys. Procedia **33**, 1986–1991 (2012)
68. Abhishek, S., Sugumaran, V., Devasenapati, S.B.: Misfire detection in an IC engine using vibration signal and decision tree algorithms. Measurement **50**, 370–380 (2014)
69. Izydorczyk, B., Wojciechowski, B.: Differential diagnosis of eating disorders with the use of classification trees (decision algorithm). Arch. Psychiatry Psychother. **18**(4), 53–62 (2016)
70. Breiman, L.: Random forests. Mach. Learn. **45**(1), 5–32 (2001). https://doi.org/10.1023/a:1010933404324
71. Geurts, P., Ernst, D., Wehenkel, L.: Extremely randomized trees. Mach. Learn. **63**(1), 3–42 (2006)
72. Bramer, M.: Nearest neighbour classification. In: Principles of Data Mining. pp. 31–38. Springer, London (2007)
73. Pan, Z., Wang, Y., Ku, W.: A new general nearest neighbor classification based on the mutual neighborhood information. Knowl.-Based Syst. **121**, 142–152 (2017)
74. Friedman, J.H.: Greedy function approximation: a gradient boosting machine. Ann. Stat. **29**(5), 1189–1232 (2001)
75. Zhu, J., Zou, H., Rosset, S., Hastie, T.: Multi-class adaboost. Stat. Interface **2**, 349–360 (2009)
76. Yelp: Yelp dataset challenge: Round 13. https://www.yelp.com/dataset/challenge (2019). Accessed Dec 27 2019
77. Scikit-learn: API Reference. https://scikit-learn.org/stable/modules/classes.html (2019). Accessed Mar 19 2019

Gregorius Satia Budhi is currently a Ph.D. candidate with the School of Electrical Engineering and Computing, The University of Newcastle, Australia. He is also an academic staff member of Petra Christian University in Indonesia. His research interests include sentiment analysis and machine learning prediction.

Raymond Chiong is a tenured academic staff member at the University of Newcastle, Australia. He is also a guest research professor with the Centre for Modern Information Management at Huazhong University of Science and Technology, Wuhan, China, a Minjiang Scholar with the School of Economics and Management at Fuzhou University, Fuzhou, China, and a visiting scholar with the Department of Automation, Tsinghua University, Beijing, China. His research interests include data analytics, optimisation, evolutionary game theory, and modelling of complex adaptive systems. He has published over 180 papers in these areas. He is the Editor-in-Chief of the Journal of Systems and Information Technology, an Editor of Engineering Applications of Artificial Intelligence, and an Associate Editor of the IEEE Computational Intelligence Magazine.

Sandeep Dhakal obtained his B.Sc. from Swinburne University of Technology, Australia, and his M.Sc. degree from the University of St. Andrews, Scotland. He is currently a Ph.D. student at the University of Newcastle, Australia. His research interests include modelling of complex adaptive systems, evolutionary game theory, and optical character recognition. He has more than 10 publications to date in these areas, and has reviewed papers for the IEEE Computational Intelligence Magazine, IEEE Transactions on Evolutionary Computation, and Engineering Applications of Artificial Intelligence.