A Non-Canonical Hybrid Metaheuristic Approach to Adaptive Data Stream Classification

Hossein Ghomeshi[a], Mohamed Medhat Gaber[a], Yevgeniya Kovalchuk[a]

[a]*School of Computing and Digital Technology, Birmingham City University, Birmingham, United Kingdom*

Abstract

Data stream classification techniques have been playing an important role in big data analytics recently due to their diverse applications (e.g. fraud and intrusion detection, forecasting and healthcare monitoring systems) and the growing number of real-world data stream generators (e.g. IoT devices and sensors, websites and social network feeds). Streaming data is often prone to evolution over time. In this context, the main challenge for computational models is to adapt to changes, known as concept drifts, using data mining and optimisation techniques. We present a novel ensemble technique called RED-PSO that seamlessly adapts to different concept drifts in non-stationary data stream classification tasks. RED-PSO is based on a three-layer architecture to produce classification *types* of different size, each created by randomly selecting a certain percentage of features from a pool of features of the target data stream. An evolutionary algorithm, namely, Replicator Dynamics (RD), is used to seamlessly adapt to different concept drifts; it allows good performing *types* to grow and poor performing ones to shrink in size. In addition, the selected feature combinations in all classification types are optimised using a non-canonical version of the Particle Swarm Optimisation (PSO) technique for each layer individually. PSO allows the types in each layer to go towards local (within the same type)

[☆]Fully documented templates are available in the elsarticle package on CTAN.
Email addresses: `Hossein.ghomeshi@mail.bcu.ac.uk` (Hossein Ghomeshi), `Mohamed.Gaber@bcu.ac.uk` (Mohamed Medhat Gaber), `Yevgeniya.Kovalchuk@bcu.ac.uk` (Yevgeniya Kovalchuk)

and global (in all types) optimums with a specified velocity. A set of experiments are conducted to compare the performance of the proposed method to state-of-the-art algorithms using real-world and synthetic data streams in immediate and delayed prequential evaluation settings. The results show a favourable performance of our method in different environments.

Keywords: Ensemble Learning, Data Stream Mining, Concept Drifts, Bio-Inspired Algorithms, Non-stationary Environments, Particle Swarm Optimisation, Replicator Dynamics.

1. Introduction

As the digital world advances, the number of data streams produced by various sources such as IoT devices and sensors and social media networks grows rapidly. Such streams are usually characterised by high velocity and changes in data distributions over time. Therefore, a significant number of recent research are concentrated on data stream classification challenges specially in non-stationary data streams [1]. The main challenge in this context is adaptation to different *concept drifts*; that is, when the data distribution evolve over time in unforeseen ways.

Various forms of concept drifts can be categorised into four generic groups of sudden (abrupt), incremental, gradual and recurrent. In sudden concept drifts, the distribution of data is suddenly replaced with a new distribution. An incremental concept drift is a drift to the distribution of data when it goes through various, unstable distributions before being stable at a specific data distribution. In the gradual concept drifts, the ratio of a new distribution of incoming data raises, and the ratio of the data from the former distribution drops over time. In a recurrent concept drift, the same old distribution of data reappears after passing some time with different distribution/s.

Ensemble learning technique is a machine learning approach, where multiple classifiers are created and combined using a voting mechanism to establish a single class as the output of a data instance. Ensemble techniques have demonstrated superiority over other classification techniques for stream classification tasks in non-stationary environments [2][3]. This is due to their flexibility in training and updating classifiers. Furthermore, ensemble techniques offer a solution to keep the effects of old and new instances using multiple classifiers since each instance in a data stream is processed only once on the arrival in most cases.

An ideal approach to non-stationary data stream classification should satisfy the following objective with a multifold features: having the least possible misclassification rate while minimising the computational complexity and quickly adapting to possible concept drifts. However, there is a lack of comprehensive approaches offering these features in a single framework; the majority of the existing ensemble methods are focused on either one or two of them, or a specific type of data streams.

The aim of this study is to propose a novel method using a modified bio-inspired algorithm, namely, Particle Swarm Optimisation (PSO), to comply with the aforementioned characteristics of an ideal approach to cope with evolving data streams in classification challenges. The proposed technique comprises a three-layer architecture. Each layer is initially assigned some predefined classification *types* that randomly created from a pool of features of the target data stream. We used an evolutionary algorithm called Replicator Dynamics (RD) to seamlessly cope with smooth (i.e. gradual or incremental) concept drifts; it allows the classification types with good performance to grow and those with poor performance to shrink in size. The combination of features in all types is then optimised using a modified version of PSO for each layer individually. This helps the method to cope with more sudden (i.e. recurring or abrupt) concept drifts. PSO allows the types in each layer to go towards local (within the same type) and global (in all types) optimums with a specified velocity.

The proposed method in this paper is evaluated over 5 real-world data streams and 4 synthetic (artificial) data stream generators. Different types of concept drifts are added to synthetic datasets in order to examine how the proposed method adapts to different concept drifts compared to state-of-the-art methods. As the process of labelling instances in the real-world datasets is different depending on the application, real-world data streams can be categorised into two different labelling mechanisms of complete labelling and partial labelling. Complete labelling is done where the true labels of respective instances are completely accessible either instantly or after a delay with no/small extra overhead to the system. This is the case in most of the forecasting tasks such as weather forecasting, stock market analysis, forest monitoring, airline predictions, and bill estimation. Partial labelling is done where the real labels are accessible with an extra overhead to the system via a third party (usually human). This is the case where the real labels are retrieved after an analysis of related data in the stream (e.g. medical diagnosis in health data, anomaly/fraud detection in credit card transactions, etc.). The main goal of the proposed framework is to deal with complete labelling data streams. Hence in the experiments part of this paper, it is assumed that the true labels of instances are completely accessible either instantly or after a specified delay. In this regards, each data-set in the experiments are processed twice; once in the delayed setting (where the actual labels of instances are accessible after a specified time) as well as once in the immediate setting (where the actual labels are accessible instantly).

The rest of this paper is organised as follows: Section 2 overviews related research; Section 3 details our proposed method; Section 4 outlines the experimental setup and results, and compares the presented RED-PSO method to other state-of-the-art methods; Section 5 overviews conclusion and outlines the possible future work.

2. Related Work

A majority of the existing data stream classification algorithms for evolving environments use ensemble learning techniques [1][2][3][4] due to their flexibility in updating the classification model (i.e. adding, removing and retraining their constituent classifiers), and consequently the fact that such methods are more trustworthy than single-classifier methods, especially in non-stationary environments, thanks to their experimentally validated higher accuracy.

In general, many of the stat-of-the-art ensemble learning algorithms are adapted versions of bagging [5] and boosting [6] methods. OzaBag [7] is an online version of the standard bagging, that every classifier in the pool is trained with k copies of the data that has received recently. OzaBoost [7] is the online version of the standard boosting algorithm. In OzaBoost method, each incoming instance is used to train all experts sequentially: highest possible weight is assigned to the first decision tree and the weights calculated for the next decision trees are based on the evaluation of the older ones. OSBoost [8] is a method based on online boosting. This method combines hypotheses from weak learners in order to obtain an effective online boosting.

Dynamic Weighted Majority (DWM) [9] maintains an ensemble of classifiers using a weighted-majority vote mechanism. DWM creates and deletes classifiers dynamically in order to response to concept drifts. In case of a classifier miss-classifying an instance, its weight is decreased by a specific value disregarding to the ensemble's output. The classifiers that have the weights less than a specified threshold are then eliminated from the ensemble. In this method, a new classifier will be created and added to the ensemble once the ensemble misclassifies an instance. Accuracy Updated Ensemble (AUE) [10] is an approach that extends Accuracy Weighted Ensemble (AWE) [11] method which is a block based ensemble approach with classifiers weighted based on their expected classification accuracy. AUE method incrementally trains old classifiers using Hoeffding trees [12] and weights them regarding to their error rate. The online version of this method is later proposed called Online Accuracy Updated Ensemble (OAUE) [13]. Anticipative Dynamic Adaptation to Concept Changes (ADACC) [14] aims to optimise the stability of the ensemble by identifying incoming concept drifts using an enhanced forgetting mechanism.

Adaptable Diversity-based Online Boosting (ADOB) [15] is a modied version of the online boosting [7] aims to speed up the recovery of classifiers after concept drifts. It uses ADaptive WINdowing (ADWIN) [16] change detector as its concept drift detector. This algorithm changes the Poisson distribution parameter from a fixed value of 1 to an adjusted value of λ according to the accuracy of its base classifiers – so that the samples can be distributed efficiently among its base classifiers.

The aforementioned methods can be categorised as implicit methods as they implicitly cope with concept drifts, while no concept drift detection mechanism is used. This is opposed to explicit methods that use a mechanism to detect concept drifts and react to them immediately (e.g. by resetting the pool of classifiers). The main issue with the implicit methods is that adaptation to a new concept may take a long time in most cases due to their implicit behaviour. Furthermore, concept drifts are not identified immediately in such approaches.

The following algorithms are some of the established explicit methods in the literature. Adaptive Boosting (Aboost) [17] algorithm updates all classifiers' weights regarding to whether or not an instance is classified correctly. Once the concept drift detector alarms, the weight of all classifiers in the ensemble is reset to one. Adwin Bagging (AdwinBag) [18] is an explicit approach based on Oza's online bagging algorithm [7]. It uses the same learning mechanism as OzaBagging method and a concept drift detector called ADaptive WINdowing (ADWIN) [16] to identify when a new classifier is needed. This algorithm is further improved in Leveraging Bagging (LevBag) [19] that aims to increase the amount of re-sampling in the bagging technique.

Adaptive Random Forest (ARF) [20] is another explicit technique that adapts the classical Random Forest algorithm [21]. It grows decision trees by training them on re-sampled versions of the original data and by randomly selecting a small number of features that can be inspected at each node for split.

The main disadvantage of the explicit methods is that such algorithms are sensitive to false alarms (noise), which leads to a reduced accuracy due to wrongly detected concept drifts. Moreover, choosing a proper concept drift detector is a difficult task. To address it, RD is employed in our proposed method offering a seamless yet effective approach to improve the performance by growing and shrinking classification *types*. On the other hand, the main disadvantage of the implicit algorithms is their slow reaction to concept drifts. We tackle this issue by using a bio-inspired mechanism to optimise the combination of classification types in each layer of the presented framework.

3. RED-PSO Framework

In this section, we present RED-PSO, a novel ensemble learning framework for non-stationary classification using the RD and PSO techniques. We train an ensemble of classifiers comprising three categories (layers) of different classification *types* generated by randomly selecting features (subspaces) from the set of attributes of the target dataset. For each layer of the proposed ensemble, the number of classification types and features in each one of them is chosen so that the higher the number of features in each classification type in a layer, the lower the number of classification types is necessary for this layer. As a result, the first layer consists of a smaller number of classification types and a bigger number of features in each classification type, whereas the third layer contains a bigger number of classification types with a smaller number of features in each one of them.

To make the proposed method effective specially in evolving environments and to a seamless adaptation to the most recent types of data distribution, we apply RD to all layers of the ensemble to grow the well-performing types and shrink the poor-performing ones in size. As the original RD is designed specifically for static datasets, we modify it to be compatible with streaming data. In addition, the randomly drawn *classification types* (subspaces) are optimised using a modified version of PSO, which is applied to each layer individually to ensure fast adaptation to different concept drifts.

3.1. Replicator Dynamics: Overview

RD is a simple model of evolution and prestige-biased learning in game theory [22][23]. This model is a solution for selecting suitable *types* to grow and unsuitable *types* to shrink among a population of diverse *types.* The act of selection in this model happens at discrete times and "the population of each *type* in the next selection is given by the replicator equation as a function of the *type's* payoff and its current proportion in the population" [24]. In general format of this methods, *types* performing better than average payoff increase in size, whereas those that perform worse than the average payoff decrease in size. The *Replicator Equation* is represented by the following formula:

$$\dot{x_i} = x_i[(Wx)_i - x^T Wx], \tag{1}$$

where $(Wx)_i$ denotes the expected payoff for an individual and $x^T Wx$ denotes the average payoff in the population state x.

In the presented method, a classification *type* is a random subspace of the total number of features of the target data stream. A *type's* payoff is the average accuracy of the classifiers inside the same *type*, and the expected payoff is the average accuracy of all existing classifiers.

3.2. Particle Swarm Optimisation: Overview

PSO is a metaheuristic algorithm [25] inspired by the social behaviour of the movement of organisms in a bird flock or fish school. The main target of the PSO algorithm is finding the global minimum of a function. While PSO does not guarantee an optimal solution, it is shown to have promising results in various applications [26].

A typical PSO algorithm is initialised by creating an initial random population (swarm) of candidate solutions (particles). The particles then move around the search space with a dynamic velocity (according to a specific formulae) to find the best possible solution. The movement of the particles is directed by two factors called 'previous best' (pbest) and 'global best' (gbest) positions and being updated from one iteration to another. The pbest position is a particle's best position throughout the history, while the gbest position is the best position achieved by all particles in the swarm. This process is repeated at each iteration – so that a satisfactory solution is discovered. The velocity (V) and position (P) of particles are updated according to the following formulas:

$$V_i(t+1) = \omega V_i(t) + r_1(pbest(i,t) - P_i(t)) + r_2(gbest(t) - P_i(t)), \tag{2}$$

$$P_i(t+1) = P_i(t) + V_i(t+1), \tag{3}$$

where ω denotes the inertia weight used to balance the global and local exploitation, and r_1 and r_2 denote positive constant parameters called acceleration coefficients.

A canonical PSO algorithm is designed to iterate over a static data, where there is only one possible optimal solution. In contract, in data stream classification tasks, data comes in an online manner and optimal solution is subject to change with time. Therefore, a non-canonical version of PSO is proposed in Section 3.3 to make PSO able to work in a streaming environment.

In the presented RED-PSO method, the initial population is drawn randomly as different subspace of features (*types*), and each layer is optimised individually using the modified version of PSO.

3.3. RED-PSO System

First, we create three different layers in a way that each layer contains a specific amount of *types* and each classification type covers a specific percentage of features from a predefined pool of features. There is a negative linear correlation between the percentage of features in each *type* and the number of *types* in each layer. In other words, the bigger the number of *types* inside each layer is, the smaller the number of features each *type* is required to cover in the same layer, which can be represented as

$$\frac{m}{f} = 1 - \frac{n}{f}, \tag{4}$$

where m denotes the total number of *types*, n denotes the total number of features that are to be selected for each *type* and f denotes the total number of features of the target data stream.

Once the parameters of each layer are specified, n_l features (attributes) are randomly selected from the set of all attributes, where n_l is n^{th} parameter for layer number l (l=1, 2, or 3). This step is repeated m_l times for each layer, where m_l is m^{th} parameter for the layer number l. As a result, we have m_l independent *types* for each layer at the end of this step.

When all layers are created using a random subspace of features in each layer, the ensemble starts the training phase using RD (see Section 3.1). In the original RD, the number of trees that need to be added to a subspace is specified dynamically as

$$T_a(i) = \lfloor (a(t_i) - \frac{\sum_{i=1}^{m} a(t_i)}{m}) \times T_n(i) \rfloor \tag{5}$$

$$T_r(i) = \lfloor (\frac{\sum_{i=1}^{m} a(t_i)}{m} - a(t_i)) \times T_n(i) \rfloor \tag{6}$$

where $T_a(i)$ denotes the number of trees to add, $T_r(i)$ denotes the number of trees to remove, $a(t_i)$ denotes the accuracy of subspace i being processed, m denotes the total number of *types* and $T_n(i)$ denotes the total number of trees currently inside subspace i. When the number of trees to be added to a subspace is more than one, different trees are created using the bootstrap aggregating (bagging) model. However, prequential evaluation is impossible in this case (i.e. when incoming data are used for testing and then training) since only one tree can be created at each time step. Hence, instead of creating different trees in a subspace at each time step, only one tree is created but with a higher weight assigned to it for the voting purpose. Therefore, a tree with $weight = 2$ has two votes in the voting mechanism (i.e. higher impact) instead of only one. The removing mechanism in this strategy is based on the performance of the classifiers; e.g. when the number of trees to remove is 2, two least accurate trees (in the last data block) are eliminated from the ensemble.

Once the first data block is collected by the ensemble, a classifier will be built for every *type* (subspace) inside all the layers. Given *max* as the maximum number of decision trees for each *type*, this step is repeated for the first $\frac{max}{2}$ blocks of data to shape the *types* and allow them to reach a specific maturity level. This phase is called *initial training* and is performed to build an average number of classifiers for every *type* in the ensemble. Note that for every data block that is received by the ensemble, all decision trees classify the instances, and the majority voting is applied to determine each layer's output. Then the ensemble's output is determined by combining the output of each layer using their weights obtained according to Equation 7. This phase is called the *voting* step.

$$W_i = W_{i-1} + \alpha(P_{i-1} - A_{i-1}), \tag{7}$$

where W_{i-1} denotes the weight of a layer at $(i-1)^{th}$ data block, P_{i-1} denotes the accuracy of the same layer over $(i-1)^{th}$ data block, A_{i-1} denotes the average accuracy of all layers over $(i-1)^{th}$ data block and α denotes the coefficient of recent data, which is an arbitrary parameter of the proposed algorithm ($\alpha > 1$).

Once the initial training phase is completed, each decision tree in a *type* is evaluated after classifying incoming instances by calculating its accuracy:

$$a_i = \frac{c_i}{db}, \tag{8}$$

where c_i denotes the number of correctly classified instances in i^{th} data block and *db* denotes the total number of instances in each data block. Accuracy of each *type* is the average accuracy of its constituent decision trees. Accuracy of the whole ensemble can be determined similarly to Equation 8. This phase is called *evaluation*.

Next, the *RD* step is applied (see Algorithm 1), where each *type's* accuracy is taken into consideration and assessed with the expected payoff (as explained in Section 3.1) calculated for each layer individually:

$$\begin{cases} a(t_i) \geq \frac{\sum_{i=1}^{m} a(t_i)}{m} \Rightarrow grow \\ \\ a(t_i) < \frac{\sum_{i=1}^{m} a(t_i)}{m} \Rightarrow shrink \end{cases}, \tag{9}$$

where $a(t_i)$ denotes the accuracy of i^{th} *type* and m denotes the total number of *types*.

The expected payoff in this study is set to the average accuracy of all *types* in each layer. Algorithm 1 shows how the *RD* step works. In this algorithm, t_j

denotes the j^{th} *type* of the ensemble ($1 \leq j \leq m$) and $a(t_j)$ denotes the accuracy of this *type*. The following functions are used in the presented algorithm:

- *Classify()*: classifying data using the majority voting;
- *Evaluate()*: evaluating the accuracy of classifiers/*types* in the ensemble using Equation 8;
- *Grow()*: adding a new classifier (decision tree) to the specified *type* (if Equation 9 stands);
- *Shrink()*: removing the worst performing one classifier (decision tree) from the specified classification *type* (if Equation 9 stands); if this *type* has only one classifier, then do nothing;
- *Train()*: training all classifiers of the ensemble by the newly received.

To set a boundary for the number of classifiers (decision trees) inside the framework, an arbitrary upper bound of $max = 10$ has been set for all the *types* in the ensemble. Hence, once the number of classifiers of a *type* is exceeded, the weakest performing decision tree of the same *type* is removed in order to make room for the newly built classifier. Furthermore, to prevent the *types* from complete removal, a lower bound of $min = 1$ is assigned to all *types*.

3.3.1. PSO Optimisation

We use a non-canonical version of PSO algorithm to optimise the combination of features of the *types* inside each layer. Initially, PSO takes all randomly drawn *types* as its input and tries to move them towards the global best (*gbest*) and local best (*pbest*) solution *types* with a specified velocity in each iteration (when a new data block is received). In this method, by term ‘moving a particle to a specific space with a velocity’, we mean to reform the combination of features of a particle (classification *type*) to resemble a specific subspace of features with a defined proportion (velocity).

Let the *gbest* subspace include a set of features G, *pbest* include a set of features L, and the current subspace include a set of features T. Then, the

Algorithm 1: Modified RD Algorithm

```
Input: Continuous block of data $DB = \{db_1, db_2, .., db_n\}$
$n$: number of features to be selected in each type
$m$: total number of types
$t_l$: types inside $l^{th}$ layer
$max$: maximum number of classifiers in each type.
Output: Updated state of classification types
1  $l := 1$
2  for $t_l := 1$ to $t_l := m$ do
3      Randomly select $n$ features
4  $i := 1$
5  while data stream is not empty do
6      if $i \leq \frac{max}{2}$ then
7          Classify($db_i$)
8          Grow(T) for all types
9      else
10         Classify($db_i$)
11         Evaluate()
12         if $a(t_j) \geq \frac{\sum_{j=1}^{n} a(t_j)}{m}$ then
13             Grow($t_j$) /* As in Equation 5 */
14         else
15             Shrink($t_j$) /* As in Equation 6 */
16     Train()
17     $i := i + 1$
```

space that a *type* can move to can be in $S = [(G - T) \cap (L - T)]$. This set will have the features in both good *types* that are not in the current one (i.e. S is the set of features the current particle can add to resemble (1) the best performing previous location of the same particle, and (2) the best performing particle in the whole swarm at the present). If the space is empty, the particle will move to space $S = [(G-T) \cup (L-T)]$. To compensate for any newly added features from S, the algorithm removes set $B = [T - (G \cap L)]$ of features in T that are not among the good particles and may have contributed to the lower performance (i.e. B is the set of features that are not in both well performing particles, and thus may be irrelevant to the current concept). Using the average of the differences in accuracy between the two good particles and the current one, we apply the move as the velocity – so that if the difference (distance) is greater than $x\%$, we can move $\beta \times 100\%$, where $\beta (0 < \beta \leq 1)$ is a constant value (coefficient) that is different for each layer and has been specified to add more diversity, especially to higher layers. If the distance is less than $x\%$, we move according to how far the distance (the proportion) is from the $x\%$ ($\beta \times \frac{d}{x}$), where d is the distance of each particle and the average of *gbest* and its *pbest* (global best and previous best) particles. If $|S| > |B|$, we use only $|B|$ out of $|S|$, just to maintain the same number of features in a *type* (i.e. when the number of features that can be added $|S|$ is greater than the number of features that can be removed $|B|$, $|B|$ is set to be the upper bound on the number of features to be added, maintaining the same number of features from one iteration to the next). In this study, the values for constant β are arbitrarily assigned as 1, 0.7 and 0.4 for layers 3, 2 and 1, respectively. Hence, the maximum velocity in layer 3 that has the lowest number of features in each *type* is set to 100%. Similarly, the maximum velocities for layers 2 and 1 are set to 70% and 40%, respectively.

Algorithm 2 shows how the modified PSO stage works. The following functions are used in this algorithm along with the ones previously introduced in this section:

- *Update()*: update the weight of each layer according to Equation 7;

- *Update-optimal()*: update the layer's global optimal *gbest* and each type's previous optimal (*pbest*) using the evaluated accuracy for each of the *types*;
- *Move()*: moving each *type* inside the layer towards its *gbest* and *pbest* with a specified velocity based on its distance (in accuracy) of each *type* to the average accuracy of optima (as explained earlier in this section);

PSO is performed in each layer of the proposed algorithm individually and is iterated in every data block received by the ensemble. As a result, the particles (*types*) inside each layer of the ensemble move towards *gbest* and *pbest* of the same layer with the specified velocity that is calculated based on their performance over the last data block, in which their true labels are known. Note that each iteration in Algorithm 2 starts only after the same data block is processed by Algorithm 1.

General ideas of RED-PSO are illustrated in Figure 1. The left triangle in the figure shows *types* (closed curves) in each layer and features (dots) in each *type*. The right triangle shows their corresponding *types* (particles) that move towards a specific particle ($gbest \cap pbest$) with a specific velocity (arrows). The smaller the classification types are (less features), the faster they move towards optimums (higher velocity).

The proposed method uses an evolutionary method (RD) to seamlessly adapt to the concept drifts that are more smooth in nature (such as gradual and incremental concept drifts) by increasing the size of good performing *types* and reducing the number of trees of poor performing *types*. At the same time, it uses a modified version of PSO to optimise the combination of features in each layer of the proposed algorithm individually, which is suitable to cope with more immediate concept drifts (such as abrupt and recurring concept drifts) that usually take longer to adapt to using the state-of-the-art methods. Finally, having three different layers of different *types* built using different parameters, such as the number of *types* (m) and number of features (n) in each *type*, can help the algorithm to be less sensitive to drifting features by minimising their effects on each layer.

3.4. Computational Complexity

Assuming the number of classes c, the number of features in each classification *type* p, the maximum number of values per feature v, and the maximum number of trees in the ensemble k, no more than p features are considered in a single Hoeffding tree [12]. Each feature at a node requires computing v values. Since calculating information gain requires c arithmetic operations, the cost of k Hoeffding trees at each time-step (iteration) in the worst case scenario is $\mathrm{O}(kcpv)$. Given the number of all classification *types* in the ensemble m and the fact that RD uses m arithmetic operations to calculate payoffs, the cost of applying RD to the ensemble is only $\mathrm{O}(m)$. Hence, the computational complexity of RED1 and RED2 variations (deploying Hoeffding trees along with RD in the proposed framework) is $\mathrm{O}(m + (kcvp))$. Furthermore, assuming that the number of classification *types* in each layer of the ensemble is m_i $(1 \leq i \leq 3)$ and the number of iterations for each data block is 1, the PSO optimisation procedure uses $m_1 + m_2 + m_3$ arithmetic operations in each time-step. Therefore the computational complexity RED-PSO variations of the proposed framework is $\mathrm{O}(m + (kcvp) + m_1 + m_2 + m_3)$.

In summary, the procedure of reading, processing and classifying data in RED-PSO is as follows. Once a new block of data has been filled up to its full capacity ($d = 1,000$ instances in this case), the system starts reading and classifying the instances in a synchronous manner using all current classifiers inside the framework. Since each processing layer of this method has a specific weight assigned to its classifiers, the majority voting takes place to determine the predicted class of each instance. Once the real labels of all instances in a data block are received, the classifiers inside each layer start moving based on the local and global optimum values according to the optimisation procedure described above. Finally, in the evaluation step, the weights of all layers are updated based on their performance over the last data block.

Algorithm 2: Modified PSO Optimisation

```
Input: Continuous block of data $DB$ ={$db_1$,$db_2$,..,$db_n$}
Randomly drawn subspaces (types) for each layer, $T_l$ ={$t_1$,$t_2$,..,$t_m$}
$W$ ={$w_1$,$w_2$,$w_3$} /* initial weight for each layer*/
Output: New set of types, $T_{l'}$ ={$t_{1'}$,$t_{2'}$,..,$t_{m'}$}
1  $i := 1$ while data stream is not empty do
2      Classify($db_i$)
3      for $l := 1$ to $l := 3$ do
4          Evaluate($T_l$)
5          Update-optimal($gbest$)
6          for $j := 1$ to $j := m$ do
7              Evaluate($t_j$)
8              Update-optimal($pbest$)
9              Move($t_j$) /*move to gbest/pbest according to its velocity */
10         Update($w_l$) /*update the weight of each layer*/
```

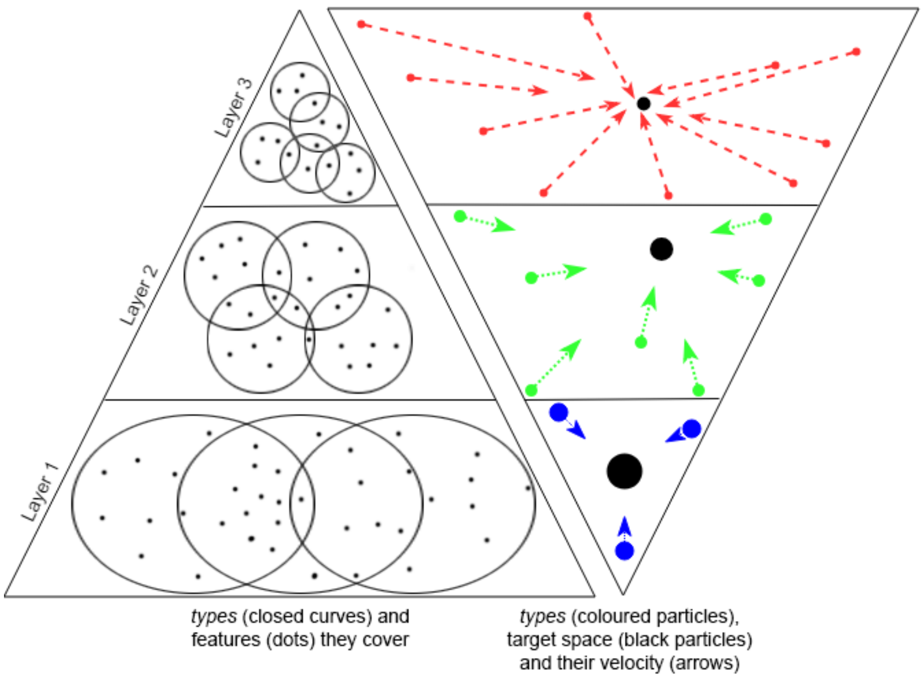

Layer 3
Layer 2
Layer 1
types (closed curves) and features (dots) they cover
types (coloured particles), target space (black particles) and their velocity (arrows)

Figure 1: Illustration of the RED-PSO algorithm using three layers. The left triangle refers to the extent of classification *types* and the features they cover in each layer. The right triangle refers to how fast each classification type would move towards global and local optimums. The smaller the classification types are (less features), the faster they move towards optimums (higher velocity).

4. Experimental Study

A collection of experiments is conducted using five real-world data streams and four synthetic data stream generators to evaluate the proposed RED-PSO framework and compare it with the existing state-of-the-art methods that have shown good performance and liable results [13][20], including Dynamic Weighted Majority (DWM) [9], Online Accuracy Updated Ensemble (OAUE) [13], OS-Boost [8], Leveraging Bag (LevBag) [19], Adaptive Random Forest (ARF) [20], Learn++.NSE [27] and Adaptable Diversity-based Online Boosting (ADOB) [15].

The proposed algorithms are developed in Java programming language, while all algorithms are executed using the Massive Online Analysis (MOA) framework [28]. When running LevBag, ARF, DWM, OAUE, OSBoost, Learn++ and ADOB, their default parameters as implemented in MOA are used, while the parameters of our proposed algorithms are listed in Section 4.2. In order to have a thorough set of experiments, 10 different variants (seeds) are generated for every synthetic data stream and each method is tested on all variants. The variants are generated by changing different parameters in all synthetic streams, as specified in Section 4.1. For every real world-data stream, each experiment is repeated 10 times over the same data stream.

As mentioned in Section 1, there are two different settings for evaluating methods over each dataset used in the set of experiments. The first setting involves passing one of the chosen datasets through a specific method using the prequential evaluation technique with immediate access to the real labels of the instances. This setting is called *immediate setting.* In the second setting, the real labels of the instances are revealed to the ensemble after a specified delay. This evaluation technique called *delayed setting.* In this setting, the parameter of delay is set to an arbitrary value of $1,000$; therefore, the actual label of each instance is revealed to the system after passing 1,000 instances.

For both immediate and delayed settings, the window size (width) is set to 1,000. The experiments were performed on a machine equipped with an Intel Core i7-4702MQ CPU @ 2.20GHz and 8.00 GB of installed memory (RAM).

4.1. Datasets

4.1.1. Synthetic Data Streams

SEA Generator is a synthetic data stream generator that simulates concept drifts over time [29]. It generates random points in a three-dimensional feature space, however, only the first two features are relevant to the labels. Ten different variants of SEA concepts are generated for our experiments, each including one million instances. Different concept drifts are manually added throughout the data stream. For the first five variants, two abrupt concept drifts with the width (of the concept drift change) of 1 are added at instance numbers 200K and 400K, and two recurrent concept drifts with the same width are added at instance numbers 600K and 800K. For the rest five variants, two gradual concept drifts with the width of 10,000 are added at instance numbers 200K and 400K, and two recurrent concept drifts with the same width are added at instance numbers 600K and 800K.

Hyperplane Generator is a synthetic data stream with drifting concepts based on the location of a rotating hyperplane [30]. A hyperplane in d-dimensional space is the set of points that satisfy $\sum_{i=1}^{d} w_i x_i = w_0$, where x_i is the i^{th} coordinator of point x. concept drifts in this data generator are simulated by changing the location of the hyperplane. The smoothness of drifting data can be changed by adjusting the magnitude of the changes. In this experiment, the number of classes and features are set to 2 and 10, respectively; the number of drifting features is changed from 2 to 6; and the magnitude of changes is set to 0.01 or 0.02 in each variant.

Random Tree Generator (RTG) builds a decision tree by randomly selecting features as split nodes and assigning random classes to them [31]. RTG allows customising the number of nominal and numeric features, and classes. In our experiments, the number of classes varies from 2 to 6, the number of features ranges from 10 to 18, and the random seed number is chosen to be either 1 or 2.

LED Generator is a data generator that aims to predict the digits shown on a LED display [32]. It contains 24 Boolean features, 17 of which are irrelevant and the remaining 7 correspond to each segment of a seven-segment LED display. The LED generator used in this paper simulates concept drifts by swapping its features. For our experiments, ten different variants are selected with different parameters. For the first five variants, the number of drifting features are set to 1, 2, 3, 4 and 5, respectively. For the next five variants, only the random seed is changed and the drifting features are the same as in the first five variants.

4.1.2. Real-World Data Streams

Forest Cover-type Data Stream is a real-world dataset from the UCI Machine Learning Repository[1] containing the forest cover type of 30×30 meter cells obtained from the US Forest Service (USFS) [33]. It consists of 581,012 instances and 54 features with the goal to predict the forest cover type from cartographic variables.

Electricity is a widely used dataset by [34] collected from the Australian New South Wales electricity market, where prices are not fixed but affected by demand and supply. It contains 45,312 instances with 8 features each. The target class specifies the change of the price (going up or down) according to the moving average of the last 24 hours.

Airlines is a dataset[2] with the task to predict whether a flight will be delayed given the scheduled departure information. It contains 539,383 records with 7 features (3 numeric and 4 nominal).

KDDcup99 is a dataset used in the "Third International Knowledge Discovery and Data Mining Tools Competition", which includes a wide variety of intrusions simulated in a military network environment and contains 41 features and 23 classes [35]. The competition task was to build a network intrusion detector and a predictive model capable of distinguishing between "bad" connections (intrusions or attacks) and "good" (normal) connections.

Poker-Hand dataset from the UCI Machine Learning Repository[3] consists of 1,000,000 instances and 11 features. Each record of this data stream is an example of a hand consisting of five playing cards drawn from a standard deck of 52.

4.2. RED-PSO Variations and Parameter Tuning

The general parameters for all variations of the RED-PSO framework used in the experiments are listed in Table 1, where D is the number of instances in each

[1]http://archive.ics.uci.edu/ml

[2]$http://kt.ijs.si/elena_ikonomovska/data.html$

[3]$https://archive.ics.uci.edu/ml/datasets/Poker+Hand$

Table 1: General parameters used for all variations of RED-PSO framework.

D	min	max	β_1	β_2	β_3	W_1	W_2	W_3
1000	1	10	1.0	0.7	0.4	1	2	4

data block, min/max is the minimum/maximum number of classifiers (trees) in each classification *type*, β_i is the diversity coefficient used for i^{th} layer and w_i is the initial weight assigned for i^{th} layer. Note that any special parameters designated for each variation are specified separately later in this section.

We compare five different variations of our proposed framework to evaluate the effect of its different characteristics and discuss the impact of employing different parameters.

RED1 uses the proposed three-layer architecture only with RD to grow well-performing *types* and shrink poor-performing ones; it does not use PSO. The following parameters are employed for each layer in this variation:

$Layer1 \Rightarrow m = 0.3 \times f$ and $n = 0.7 \times f$;

$Layer2 \Rightarrow m = 0.5 \times f$ and $n = 0.5 \times f$;

$Layer3 \Rightarrow m = 0.7 \times f$ and $n = 0.3 \times f$.

RED2 uses the proposed three-layer architecture only with RD to grow well-performing *types* and shrink poor-performing ones; it does not use PSO optimisation. The following parameters are employed for each layer in this variation:

$Layer1 \Rightarrow m = 0.5 \times f$ and $n = 0.5 \times f$;

$Layer2 \Rightarrow m = 0.3 \times f$ and $n = 0.7 \times f$;

$Layer3 \Rightarrow m = 0.1 \times f$ and $n = 0.9 \times f$.

RED-PSO1 uses PSO in addition to RD in RED1 variation with the following additional parameters for the threshold of the maximum velocity (x):

$x = 2\%$.

RED-PSO2 uses PSO in addition to RD in RED2 variation with the following additional parameter for the threshold of the maximum velocity (x):

$x = 4\%$.

RED-PSO3 uses PSO in addition to RD in RED2 variation with the following additional parameter for the threshold of the maximum velocity (x): $x = 2\%$.

4.3. Results and Discussion

All algorithms in our set of experiments are compared using standard criteria, including the prequential accuracy in immediate and delayed settings, Kappa M (a comparison with a majority-class classifier) and overall evaluation time (classification time and training/updating time). The first part of our set of experiments is related to comparing different variations of the proposed framework to understand the effects of using different mechanisms and parameters (such as modified versions of RD and PSO) in the RED-PSO framework.

4.3.1. Comparison of the different variations of RED-PSO

Figures 2 and 3 illustrate the average accuracy for all variations of the proposed RED-PSO framework over the nine datasets in the immediate and delayed settings using bar charts. It is clear that the PSO-optimised variations (RED-PSO1, RED-PSO2 and RED-PSO3) perform better than the RD-only variations (RED1 and RED2). This is because the former variations include an additional optimisation technique to move all *types* in each layer towards the best possible spaces (*gbest* and *pbest*) in the same layer. This enables the system to optimise the combination of features in every *type* according to the time and most recent data received by the system, especially upon concept drifts. RED-PSO3 has the best average accuracy over the Hyperplane, RTG, SEA, KDDcup, Airlines and Forest Cover-type datasets, and the best overall average accuracy in both the immediate and delayed settings.

Table 2 shows the overall evaluation time of different RED-PSO variations in the immediate setting (the variation of the evaluation times between the immediate and delayed settings is negligible). It is clear that the variations without PSO optimisation are faster than the ones with it due to the time complexity of

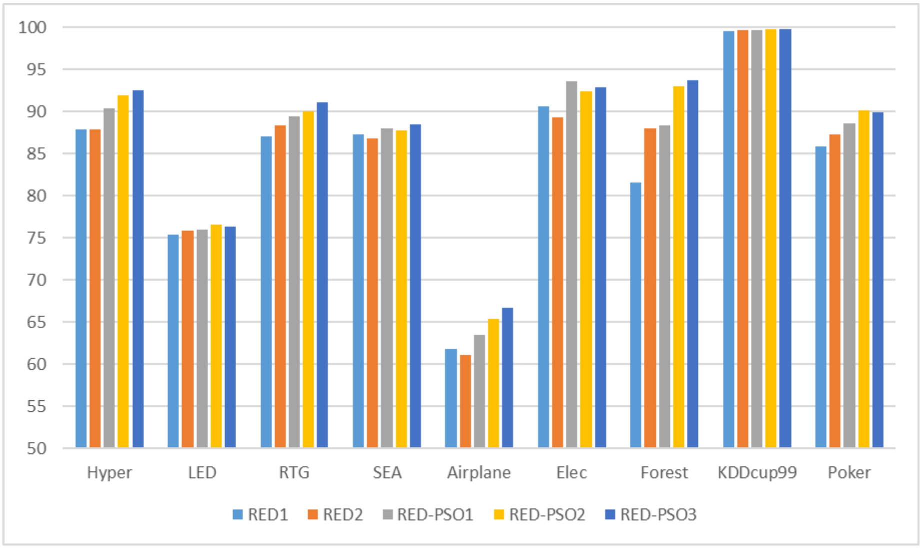


Figure 2: Average accuracy in the immediate setting for the different variations of the RED-PSO framework.

PSO being added to the ensemble. RED2 variation has the shortest evaluation time over seven out of nine datasets compared to the other variations used in the experiments, while RED-PSO2 and RED-PSO3 have the longest evaluation time for the majority of the datasets.

4.3.2. Comparison with other methods

The second part of our set of experiments involves comparing seven different state-of-the-art methods to the best performing variation of our proposed framework, namely, RED-PSO3. Figures 4 and 5 illustrate the average accuracy of RED-PSO3 compared to that of the considered state-of-the-art methods over all datasets using bar charts. It can be noticed that RED-PSO3 has the best average accuracy over the Hyperplane, LED, Airlines, Electricity and Poker-hand datasets in both the immediate and delayed settings.

Figure 6 demonstrates the overall average accuracy of the state-of-the-art methods compared to that of RED-PSO3 in both the immediate and delayed settings over all datasets. It is evident that RED-PSO3 has the best average

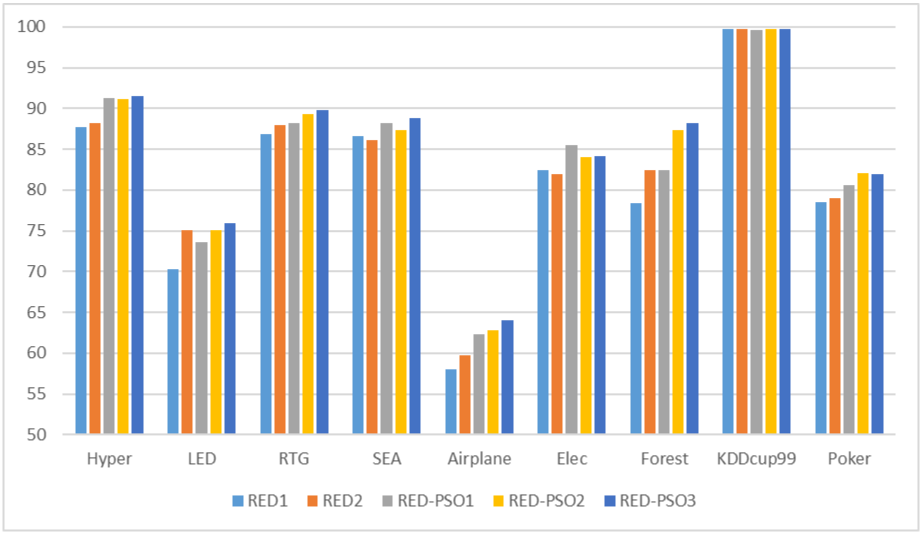


Figure 3: Average accuracy in the delayed setting for the different variations of the RED-PSO framework.

accuracy in both settings. Therefore, it can be concluded that RED-PSO3 has the most consistent performance compared to that of the other methods for different data streams. This is opposed to the ADOB algorithm that has a drastically weak performance over the LED and Forest Cover-type datasets, whereas it has the best performance over the KDDcup99 dataset. One possible explanation for the more stable performance of the proposed method is that in the RED-PSO3 framework, there are different strategies adopted for different environments and concept drifts using different layers of PSO optimisation and their dynamic weights.

The next step in our comparison is the evaluation of the classification performance according to $Kappa_M$ statistic, which is a robust inter-rater agreement for qualitative items. Table 3 lists the average $Kappa_M$ values for RED-PSO3 and the other compared methods in the delayed setting. The proposed RED-PSO3 variation has the best performance over the Hyperplane, SEA, Airline and Poker-hand datasets and the highest average $Kappa_M$ value. Taken together, these results demonstrate that the majority of the agreements in the ensemble

Table 2: Overall evaluation time (in seconds) of executing RED-PSO variations in the immediate setting. Bold values indicate the best performance for each dataset.

Dataset	RED1	RED2	RED-PSO1	RED-PSO2	RED-PSO3
Hyper.	184	**172**	279	230	253
LED	204	**189**	400	379	388
RTG	281	**243**	405	358	379
SEA	**163**	176	296	320	308
Airlines	398	**365**	528	494	543
Elec.	9.3	**8.9**	17.8	16.1	16.4
Forest	741	**722**	10871	1167	1334
KDDcup	269	**229**	349	301	308
Poker	**152**	159	201	228	210

did not occur by chance.

Furthermore, Table 4 lists the evaluation time of the proposed method compared to that of the other state-of-the-art methods in the immediate setting. Since the evaluation time values in the delayed setting are similar to those in the immediate setting, respectively, we only show the results for the immediate setting. According to the experimental results, DWM and OSBoost algorithms have the shortest evaluation time by far for the majority of the datasets (DWM over the SEA, Airlines, Electricity and Poker-Hand datasets, and OSBoost over the Hyperplane and LED datasets), while the ADOB method has the longest evaluation time over four out of nine datasets (RTG, Airlines, Electricity, Forest Cover-type and Poker-Hand). According to the overall evaluation times of different methods listed in Table 4, ADOB and Learn++ are the most time consuming methods by far comparing to the other six methods (by more than 10,000 seconds). Apart from the high evaluation times of Learn++ and ADOB, the proposed method (RED-PSO3) has the longest overall evaluation time compared to that of the other methods (by about 1,000 seconds compared to the

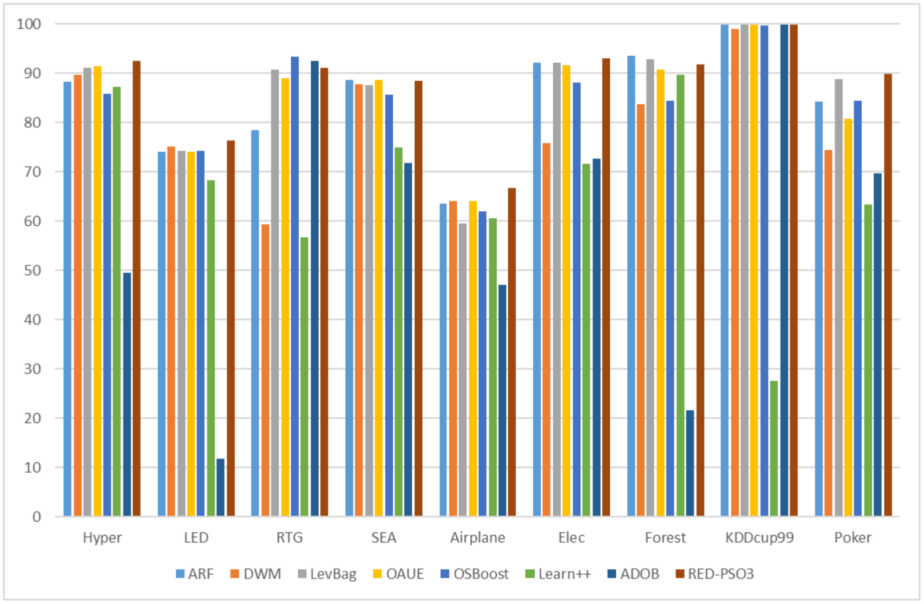


Figure 4: Average accuracy of RED-PSO3 and other state-of-the-art methods in the immediate setting.

ARF method which is the next slowest method). It seems possible that this is due to having three different layers of optimisation that run in parallel in the proposed framework. It is worth noting that given the design of our proposed classification system, the three optimisation layers can greatly benefit from parallel processing, as they operate independently when optimising the classification types. This can potentially provide a multifold speed-up of the system.

4.3.3. Performance over different types of concept drifts

We added different types of concept drifts to the datasets generated by the SEA data stream generator and compared the prequential performance of the considered algorithms in each case. Details on how different concept drifts were added to the data streams are discussed in Section 4.1. The reason for choosing a synthetic dataset for these experiments is that the exact time and actual type of concept drifts present in real-world data streams often remain unknown.

Table 3: Kappa M Statistic of different methods in the delayed setting. Bold values indicate the best performance for each dataset.

Dataset	ARF	DWM	Lev-Bag	OAUE	OS-Boost	Learn-++	ADOB	RED-PSO3
Hyper.	75.45	78.24	81.03	81.7	70.69	72.63	-4.87	**81.19**
LED	**71.03**	70.85	70.86	70.76	70.84	64.71	-2.06	70.99
RTG	66.89	35.84	85.10	81.98	76.87	40.52	**82.89**	80.98
SEA	72.01	68.57	70.81	71.95	60.55	43.78	36.61	**73.11**
Airlines	13.14	9.06	4.4	13.96	11.91	-13.60	-24.27	**15.80**
Elec.	**67.44**	30.19	60.38	57.60	55.03	7.06	26.77	65.21
Forest	46.92	0.58	35.68	**65.51**	6.72	32.25	-341.3	32.49
KDDcup	99.45	97.99	99.59	99.50	99.41	-57.12	**99.72**	99.43
Poker	24.63	-7.06	51.15	36.47	54.38	5.99	-11.77	**57.3**

Table 4: Overall evaluation time (in seconds) of executing RED-PSO3 compared to the state-of-the-art methods in the immediate setting.

Dataset	ARF	DWM	Lev-Bag	OAUE	OS-Boost	Learn-++	ADOB	RED-PSO3
Hyper.	208	130	144	107	**93**	239	298	253
LED	188	851	246	227	**174**	301	340	388
RTG	394	195	207	**148**	1141	531	1261	379
SEA	751	**98**	409	139	162	240	284	308
Airlines	495	**66**	531	366	74	977	2140	543
Elec.	7.73	**1.48**	5.12	3.05	2.06	5.9	221	16.4
Forest	153	148	206	180	114	**67**	2292	1334
KDDcup.	**56**	581	130	204	138	9819	5979	308
Poker	167	**46**	81	66	64	1720	2006	210
Overall	2419	2116	1959	1440	1942	13899	14821	3409

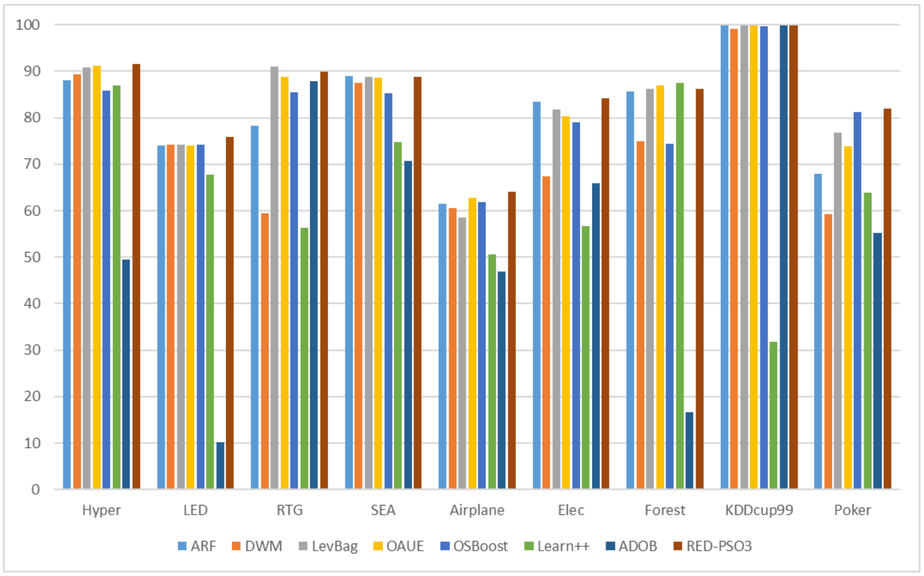


Figure 5: Average accuracy of RED-PSO3 and other state-of-the-art methods in the delayed setting

Figure 7(a) illustrates how different methods adapt to an abrupt drift with a width of 1 at an instance number of 200K. It is evident that RED-PSO3 has the highest accuracy value right after the concept drift happens. Furthermore, RED-PSO3 recovers from the introduced concept drift faster than the other methods, whereas Leverage Bagging and OSBoost fail to recover from the concept drift in a timely manner. Figure 7(b) illustrates the case when a gradual drift was added with a width of 10K. It can be noticed from the figure that all methods except OSBoost fully recovered right after the drift is over. This result suggests that in the case of gradual concept drifts, OSBoost may take a long time to have a full recovery. Figure 7(c) illustrates the behaviour of the methods upon a recurrent drift with a width of 1. Again, RED-PSO3 has the lowest accuracy drop among all other methods. At the same time, it struggles to fully recover from the drift initially. In this case, the ARF method adapts to the introduced concept drift faster than the other methods. Furthermore, ADOB and Learn++ demonstrate a drastic decrease in accuracy and fail to recover from the drift in

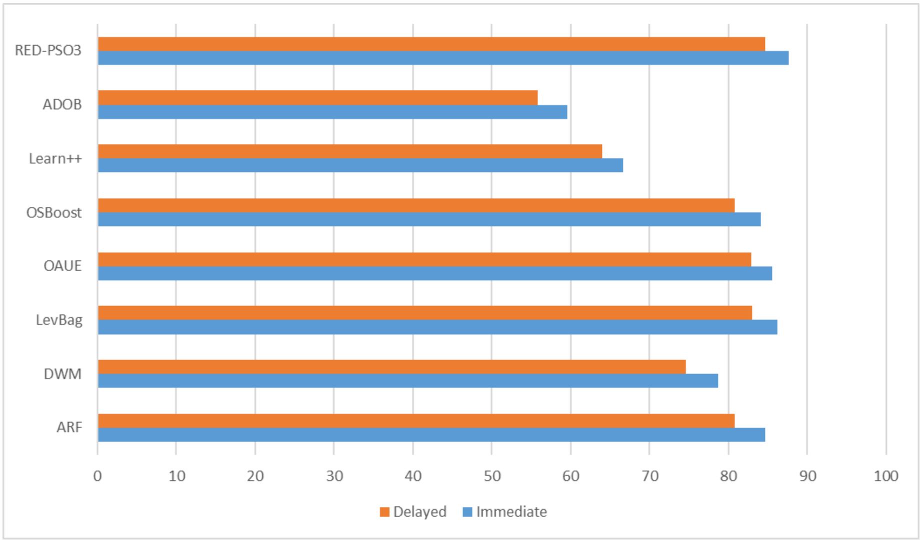
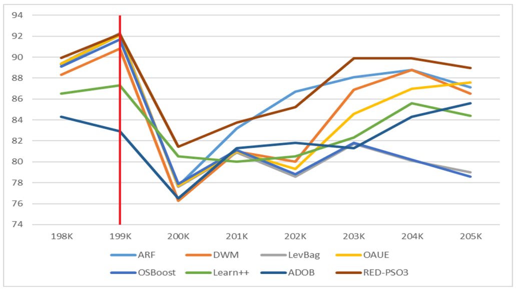


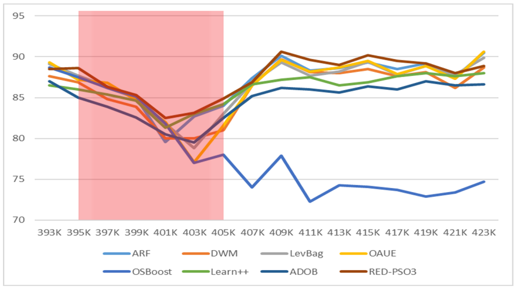
Figure 6: Average accuracy of RED-PSO3 and other state-of-the-art methods in the immediate (blue bars) and delayed (orange bars) settings over all the datasets.

a timely manner. Finally, Figure 7(d) illustrates the case when a recurrent drift was added with a width of 10K at an instance number of 795K. It is evident that the accuracy of all algorithms did not change drastically upon the drift. It is worth noting that both ADOB and Learn++ still struggled to adapt to the previous concept drift introduced at an instance number of 600K. A possible explanation may be that their concept drift detectors fail to detect the drift.

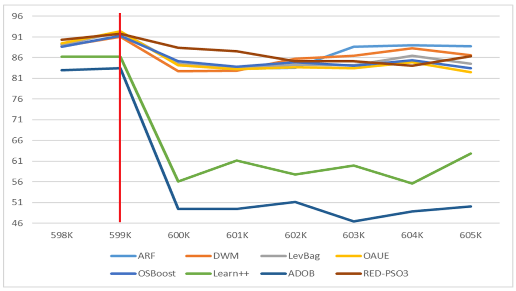
In summary, according to the experimental results, the main advantage of RED-PSO3 is its **accuracy** and **robust performance**. In particular, the proposed method demonstrated the **best average rank** and consistent performance compared to other state-of-the-art methods in both the immediate and delayed settings and upon introducing different types of concept drifts. The main drawback of RED-PSO3 is its evaluation time. While the overall evaluation time of the proposed method is not the longest among the other considered state-of-the-art methods, it is relatively long, especially over the datasets with a high number of features such as Forest Cover-type. This is due to the fact that the number of classifiers in the ensemble increases with the number of fea-



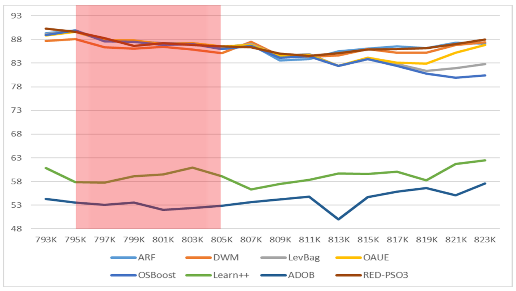
(a) Abrup Drift



(b) Gradual Drift



(c) Recurrent Abrupt Drift



(d) Recurrent Gradual Drift

Figure 7: Classification accuracy of the considered algorithms over the SEA dataset upon different types of drifts in the delayed prequential setting. The red boxes indicate the length and location of the added concept drifts.

Table 5: Average rank of the methods considered in the experiments.

Dataset	ARF	DWM	Lev-Bag	OAUE	OS-Boost	Learn-++	ADOB	RED-PSO3
R_j	3.833	5.333	3.167	3.528	4.889	6.722	6.500	**2.028**
R_j^2	14.694	28.444	10.028	12.445	23.901	45.188	42.250	4.111

tures in the target data stream. Furthermore, the adoption of an evolutionary algorithms (RD) along with a bio-inspired optimisation algorithm (PSO) in the proposed framework lead to a high computational complexity to the system. To overcome these limitations, other variations of the RED-PSO framework such as RED1 and RED2 can be used in case of high-dimensional and time-restricted applications, and also parallel processing can be applied, as aforementioned.

4.4. Statistical Analysis

The Friedman test [36] is a popular non-parametric statistical test that can be used to detect differences across several algorithms in multiple test attempts (e.g. datasets).

Table 5 shows the average rank of each algorithm considered in our experiments and their squared values with $k = 8$ and $N = 18$ since the total number of methods is eight and the total number of datasets in both the immediate and delayed setting is 18 $(9 + 9)$. Providing that the value of the Friedman test statistic is $\chi_F^2 = 57.18$ with 7 $(k - 1)$ degrees of freedom, and the critical value for the Friedman test given $k = 8$ and $N = 18$ is 18.48 at a significance level of $\alpha = 0.01$, we can conclude that the accuracy values of the studied methods are significantly different (57.18 is greater than 14.63).

Now that the Null-hypothesis is rejected, we can proceed with a post-hoc test. The Nemenyi test [37] can be used when all classifiers are compared to each other [38].

The critical value in our experiments with $k = 8$ and $\alpha = 0.10$ is $CD_{0.10} = 1.805$. As a result, the accuracy of the proposed RED-PSO3 method is significantly different from the DWM, OSBoost, Learn++ and ADOB methods, while it is not significantly different from the LevBag, OAUE and ARF methods. Figure 8 illustrates the statistical comparison of the methods considered in our experiments based on the Nemenyi test.

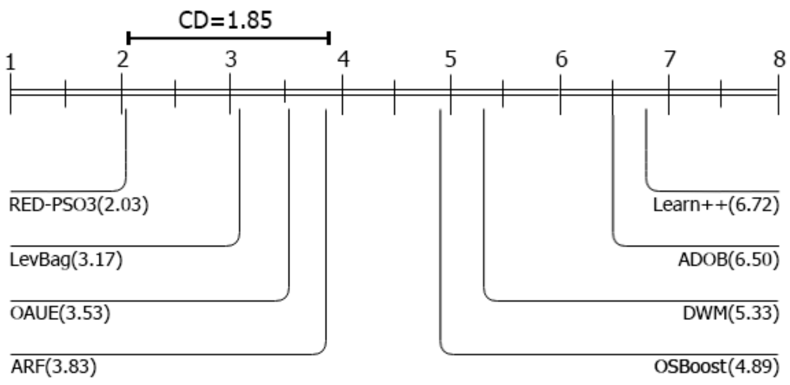


Figure 8: Comparison of all methods using Nemenyi test at $\alpha = 0.10$.

5. Conclusion and Future Work

We proposed a novel ensemble learning framework called RED-PSO to seamlessly adapt to different concept drifts in non-stationary data stream classification tasks. RED-PSO framework is based on a three-layer architecture to produce classification *types* of different size that are created by randomly selecting features from a pool of features of the target data stream. An evolutionary algorithm, Replicator Dynamics (RD), is used to seamlessly adapt to different concept drifts. Furthermore, a modified version of a bio-inspired optimisation algorithm, Particle Swarm Optimisation (PSO), is applied to optimise the combination of each *type* in all layers.

A set of experiments were conducted to compare the performance of the different variations of the proposed method, as well as the best-performing one to some state-of-the-art algorithms over five real-world and four synthetic data streams using the immediate and delayed prequential evaluation methods. According to the experimental results, RED-PSO3 has the lowest rank and highest average accuracy compared to that of the other RED-PSO variations and considered state-of-the-art methods. Using the Friedman statistical test, it was shown that the accuracy values of the studied methods were significantly different. Furthermore, according to the Nemenyi test, the accuracy of RED-PSO3 was significantly different from four out of seven compared methods (DWM, Learn++, ADOB and OSBoost), while it was not significantly different from the other three methods (LevBag, OAUE and ARF). The main drawback of the proposed framework is its long overall evaluation time in the case when the target data stream has a high number of features, which can be addressed in the future, applying parallelisation of the optimisation layers that are operating independently.

Our future plans also include analysing theoretically and testing practically the following: (1) applying other evolutionary and bio-inspired algorithms to seamlessly cope with different concept drifts; (2) introducing a new concept drift detection system based on analysing the behaviour (growing and shrinking) of the classification *types*; and (3) proposing an approach to choose the parameter for the threshold of employing the maximum allowed velocity (x) dynamically.

References

[1] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, A. Bouchachia, A survey on concept drift adaptation, ACM Computing Surveys (CSUR) 46 (4) (2014) 44.

[2] H. M. Gomes, J. P. Barddal, F. Enembreck, A. Bifet, A survey on ensemble learning for data stream classification, ACM Computing Surveys (CSUR) 50 (2) (2017) 23.

[3] B. Krawczyk, L. L. Minku, J. Gama, J. Stefanowski, M. Woźniak, Ensemble learning for data stream analysis: a survey, Information Fusion 37 (2017) 132–156.

[4] F. Chu, C. Zaniolo, Fast and light boosting for adaptive mining of data streams, in: Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer, 2004, pp. 282–292.

[5] L. Breiman, Bagging predictors, Machine learning 24 (2) (1996) 123–140.

[6] Y. Freund, R. E. Schapire, A decision-theoretic generalization of on-line learning and an application to boosting, Journal of computer and system sciences 55 (1) (1997) 119–139.

[7] N. C. Oza, Online bagging and boosting, in: Systems, man and cybernetics, 2005 IEEE international conference on, Vol. 3, IEEE, 2005, pp. 2340–2345.

[8] S.-T. Chen, H.-T. Lin, C.-J. Lu, An online boosting algorithm with theoretical justifications, arXiv preprint arXiv:1206.6422.

[9] J. Z. Kolter, M. A. Maloof, Dynamic weighted majority: An ensemble method for drifting concepts, Journal of Machine Learning Research 8 (Dec) (2007) 2755–2790.

[10] D. Brzezinski, J. Stefanowski, Reacting to different types of concept drift: The accuracy updated ensemble algorithm, IEEE Transactions on Neural Networks and Learning Systems 25 (1) (2014) 81–94.

[11] H. Wang, W. Fan, P. S. Yu, J. Han, Mining concept-drifting data streams using ensemble classifiers, in: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining, AcM, 2003, pp. 226–235.

[12] P. Domingos, G. Hulten, Mining high-speed data streams, in: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2000, pp. 71–80.

[13] D. Brzezinski, J. Stefanowski, Combining block-based and online methods in learning ensembles from concept drifting data streams, Information Sciences 265 (2014) 50–67.

[14] G. Jaber, An approach for online learning in the presence of concept change, Ph.D. thesis, Citeseer (2013).

[15] S. G. T. de Carvalho Santos, P. M. G. Júnior, G. D. dos Santos Silva, R. S. M. de Barros, Speeding up recovery from concept drifts, in: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Springer, 2014, pp. 179–194.

[16] A. Bifet, R. Gavalda, Learning from time-changing data with adaptive windowing, in: Proceedings of the 2007 SIAM International Conference on Data Mining, SIAM, 2007, pp. 443–448.

[17] F. Chu, C. Zaniolo, Fast and light boosting for adaptive mining of data streams, in: Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer, 2004, pp. 282–292.

[18] A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, R. Gavaldà, New ensemble methods for evolving data streams, in: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2009, pp. 139–148.

[19] A. Bifet, G. Holmes, B. Pfahringer, Leveraging bagging for evolving data streams, Machine Learning and Knowledge Discovery in Databases (2010) 135–150.

[20] H. M. Gomes, A. Bifet, J. Read, J. P. Barddal, F. Enembreck, B. Pfharinger, G. Holmes, T. Abdessalem, Adaptive random forests for evolving data stream classification, Machine Learning (2017) 1–27.

[21] L. Breiman, Random forests, Machine learning 45 (1) (2001) 5–32.

[22] I. M. Bomze, Lotka-volterra equation and replicator dynamics: a two-dimensional classification, Biological cybernetics 48 (3) (1983) 201–211.

[23] J. Hofbauer, K. Sigmund, Evolutionary game dynamics, Bulletin of the American Mathematical Society 40 (4) (2003) 479–519.

[24] K. Fawgreh, M. M. Gaber, E. Elyan, A replicator dynamics approach to collective feature engineering in random forests, in: Research and Development in Intelligent Systems XXXII, Springer, 2015, pp. 25–41.

[25] J. Kennedy, R. Eberhart, Particle swarm optimization, proceedings of ieee international conference on neural networks (icnn95) in (1995).

[26] R. Poli, An analysis of publications on particle swarm optimization applications, Essex, UK: Department of Computer Science, University of Essex.

[27] R. Elwell, R. Polikar, Incremental learning of concept drift in nonstationary environments, IEEE Transactions on Neural Networks 22 (10) (2011) 1517–1531.

[28] A. Bifet, G. Holmes, R. Kirkby, B. Pfahringer, Moa: Massive online analysis, Journal of Machine Learning Research 11 (May) (2010) 1601–1604.

[29] W. N. Street, Y. Kim, A streaming ensemble algorithm (sea) for large-scale classification, in: Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2001, pp. 377–382.

[30] G. Hulten, L. Spencer, P. Domingos, Mining time-changing data streams, in: Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2001, pp. 97–106.

[31] P. Domingos, G. Hulten, Mining high-speed data streams, in: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2000, pp. 71–80.

[32] L. Breiman, J. Friedman, C. J. Stone, R. A. Olshen, Classification and regression trees, CRC press, 1984.

[33] J. A. Blackard, D. J. Dean, Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables, Computers and electronics in agriculture 24 (3) (1999) 131–151.

[34] M. Harries, N. S. Wales, Splice-2 comparative evaluation: Electricity pricing.

[35] K. Cup, Data (1999), URL: http://kdd. ics. uci. edu/databases/kddcup99/kddcup99. html.

[36] M. Friedman, A comparison of alternative tests of significance for the problem of m rankings, The Annals of Mathematical Statistics 11 (1) (1940) 86–92.

[37] P. Nemenyi, Distribution-free multiple comparisons, in: Biometrics, Vol. 18, INTERNATIONAL BIOMETRIC SOC 1441 I ST, NW, SUITE 700, WASHINGTON, DC 20005-2210, 1962, p. 263.

[38] J. Demšar, Statistical comparisons of classifiers over multiple data sets, Journal of Machine learning research 7 (Jan) (2006) 1–30.

755 Supplementary Materials

Table S1: Average accuracy (%) of RED-PSO variations in the immediate setting. Bold values indicate the best performance for each dataset.

Dataset	RED1	RED2	RED-PSO1	RED-PSO2	RED-PSO3
Hyperplane	87.83	87.94	90.43	91.92	**92.54**
LED	75.41	75.88	76.01	**76.54**	76.29
RTG	87.06	88.32	89.44	89.98	**91.09**
SEA	87.30	86.78	88.01	88.34	**88.50**
Airlines	62.85	65.06	63.49	65.34	**66.68**
Electricity	90.61	89.34	**93.56**	92.43	92.86
Forest	87.56	91.01	88.42	92.99	**93.71**
KDDcup99	99.63	99.70	99.62	99.79	**99.80**
Poker	85.89	87.34	88.56	**90.10**	89.89
Overall Average	84.90	85.71	86.39	87.49	**87.93**

Table S2: Average accuracy (%) of RED-PSO variations in the delayed setting. Bold values indicate the best performance for each dataset.

Dataset	RED1	RED2	RED-PSO1	RED-PSO2	RED-PSO3
Hyperplane	87.74	88.14	91.32	91.18	**91.48**
LED	70.29	75.12	73.61	**76.03**	75.91
RTG	86.89	87.90	88.16	89.26	**89.81**
SEA	86.66	86.14	88.14	88.34	**88.80**
Airlines	60.01	62.76	61.29	63.87	**64.04**
Electricity	82.45	81.90	**85.51**	84.00	84.18
Forest	80.34	84.43	82.42	87.32	**88.19**
KDDcup99	99.73	99.70	99.60	99.76	**99.77**
Poker	78.54	79.05	80.56	**82.06**	81.98
Overall Average	81.40	82.79	83.40	84.62	**84.91**

Table S3: Accuracy (%) of the methods compared in the immediate setting. Bold values indicate the best performance for each dataset.

Dataset	Criteria	ARF	DWM	Lev-Bag	OAUE	OS-Boost	Learn-++	ADOB	RED-PSO3
Hyper.	Ave.	88.17	89.64	91.03	91.42	85.85	87.20	49.54	**92.54**
	σ	1.90	0.83	1.60	1.46	3.01	2.31	5.34	3.00
LED	Ave.	74.05	75.05	74.22	73.99	74.15	68.28	11.70	**76.29**
	σ	0.31	3.10	0.31	0.10	0.11	1.98	2.64	2.21
RTG	Ave.	78.35	59.35	90.78	88.88	**93.40**	56.68	92.4	91.09
	σ	8.12	8.87	2.26	3.26	1.45	5.32	1.36	3.36
SEA	Ave.	88.67	87.72	87.59	**88.69**	85.56	74.90	71.75	88.50
	σ	0.58	0.57	1.67	0.58	0.35	1.29	1.33	0.64
Airlines	Ave.	63.53	63.97	59.42	64.02	61.98	60.51	46.98	**66.68**
	σ	1.23	0	0.73	0	0	0	0	2.67
Elec.	Ave.	92.17	75.73	92.09	91.60	88.02	71.53	72.76	**92.96**
	σ	0.94	0	1.48	0	0	0	0	3.87
Forest	Ave.	93.57	83.75	92.73	90.70	84.45	89.69	21.59	**93.71**
	σ	1.58	0	2.10	0	0	0	0	2.67
KDDcup	Ave.	99.81	99.04	99.82	99.80	99.74	27.55	**99.88**	99.80
	σ	0.06	0	0.01	0	0	0	0	0.08
Poker	Ave.	84.19	74.37	88.52	80.74	84.31	63.41	69.64	**89.89**
	σ	4.55	0	3.34	0	0	0	0	4.35

Table S4: Accuracy (%) of the methods compared in the delayed setting. Bold values indicate the best performance for each dataset.

Dataset	Criteria	ARF	DWM	Lev-Bag	OAUE	OS-Boost	Learn-++	ADOB	RED-PSO3
Hyper.	Ave.	88.05	89.41	90.77	91.10	85.74	86.84	49.54	**91.48**
	σ	2.02	0.95	1.71	1.59	3.06	2.42	5.56	2.92
LED	Ave.	74.00	74.14	74.21	74.06	74.13	67.80	10.10	**75.91**
	σ	0.40	0.16	0.15	0.14	0.04	1.25	3.20	2.32
RTG	Ave.	78.24	59.49	**90.91**	88.72	85.53	56.28	87.88	89.81
	σ	8.06	8.67	2.48	5.13	2.90	4.69	1.21	3.79
SEA	Ave.	**88.94**	87.48	88.70	88.54	85.31	74.75	70.75	88.80
	σ	0.59	1.02	1.45	0.70	0.42	1.85	1.56	0.82
Airlines	Ave.	61.42	60.57	58.49	62.73	61.80	50.58	46.98	**64.04**
	σ	1.12	0	0.89	0	0	0	0	3.35
Elec.	Ave.	83.51	67.43	81.78	80.20%	79.04	56.60	65.81	**84.18**
	σ	1.19	0	0.88	0	0	0	0	2.81
Forest	Ave.	85.65	74.93	86.22	86.84	74.47	87.42	16.59	**88.19**
	σ	02.60	0	2.72	0	0	0	0	2.31
KDDcup	Ave.	99.80	99.12	99.81	99.78	99.74	31.68	**99.88**	99.77
	σ	0.07	0	0.01	0	0	0	0	0.08
Poker	Ave.	67.95	59.31	76.78	73.81	81.23	63.87	55.11	**81.98**
	σ	2.92	0	3.72	0	0	0	0	3.81