Support Vector Machines Trained with Evolutionary Algorithms Employing Kernel Adatron for Large Scale Classification of Protein Structures

Nancy Arana-Daniel, Alberto A. Gallegos, Carlos López-Franco, Alma Y. Alanís, Jacob Morales and Adriana López-Franco

Centro Universitario de Ciencias Exactas e Ingenieras, Universidad de Guadalajara, Guadalajara, Jalisco, México.

ABSTRACT: With the increasing power of computers, the amount of data that can be processed in small periods of time has grown exponentially, as has the importance of classifying large-scale data efficiently. Support vector machines have shown good results classifying large amounts of high-dimensional data, such as data generated by protein structure prediction, spam recognition, medical diagnosis, optical character recognition and text classiffication, etc. Most state of the art approaches for large-scale learning use traditional optimization methods, such as quadratic programming or gradient descent, which makes the use of evolutionary algorithms for training support vector machines an area to be explored. The present paper proposes an approach that is simple to implement based on evolutionary algorithms and Kernel-Adatron for solving large-scale classiffication problems, focusing on protein structure prediction. The functional properties of proteins depend upon their three-dimensional structures. Knowing the structures of proteins is crucial for biology and can lead to improvements in areas such as medicine, agriculture and biofuels.

KEYWORDS: support vector machines, evolutionary algorithms, kernel-adatron, large scale learning, machine learning, protein structure prediction

CITATION: Daniel et al. Support Vector Machines Trained with Evolutionary Algorithms Employing Kernel Adatron for Large Scale Classification of Protein Structures. *Evolutionary Bioinformatics* 2016:12 285–302 doi: 10.4137/EBO.S40912.

TYPE: Original Research

RECEIVED: September 1, 2016. **RESUBMITTED:** October 19, 2016. **ACCEPTED FOR PUBLICATION:** October 20, 2016.

ACADEMIC EDITOR: Liuyang Wang, Associate Editor

PEER REVIEW: Three peer reviewers contributed to the peer review report. Reviewers' reports totaled 679 words, excluding any confidential comments to the academic editor.

FUNDING: Authors disclose no external funding sources.

COMPETING INTERESTS: Authors disclose no potential conflicts of interest.

CORRESPONDENCE: nancyaranad@gmail.com

Introduction

With the drastic increase of data to be processed in really short amounts of time, new problems have appeared. Chromosome classification, spam filtering, defining which advertisement to show to a person on a web page, recognition of human activities and protein structure prediction are a few applications that involve immense amounts of high-dimensional data.[1,2] Sometimes the dimension and/or the number of data samples is too large, making the storage of a dataset in a computer impossible. This problem is solved by large-scale classification learning, which aims to find a function that relates the data and their corresponding class labels for an amount of data that cannot be stored in a modern computer's memory.[3] The main concern (constraint) is the amount of time that an algorithm takes to obtain an accurate result, rather than the number of samples to process.[4]

A typical problem that support vector machines (SVMs) have to face while working with a large dataset is that learning algorithms are typically quadratic and require several scans of a dataset. Three common strategies can be used to reduce this practical complexity:[3,4]

- Solving several smaller problems by working on subsets of the training data instead of the complete large dataset.
- Parallelizing the learning algorithm.
- Designing a less complex algorithm that gives an approximate solution with equivalent or superior performance.

This work presents a novel approach to solving large-scale learning problems by designing a less complex algorithm to train a large-scale SVM. Our approach uses a combination of Kernel-Adatron (KA) and some state-of-the-art evolutionary algorithms (EAs), to solve, principally, protein structure prediction (PSP) and other large-scale learning problems.[5] The obtained algorithm works with small sub-problems, has low computational complexity and is easy to implement; in addition to providing accurate generalization results, such methodology is also highly parallelizable.

Support vector machines. Since the SVM algorithm was first introduced by Vladimir Vapnik in 1995, it has been one of the most popular methods for classification because of: its simple model, the use of kernel functions and the convexity of the function to optimize (it only has a global minimum).[6] SVM's characteristics make it more appealing for classification problems with high precision requirements than other models such as multilayer perceptron, radial basis function network, Hopfield network, etc.[7–9] Many large-scale training algorithms have been proposed for SVMs with the main idea is of minimizing a regularized risk function R and maximizing the margin of separation between classes (Fig. 1) by solving Equation 1

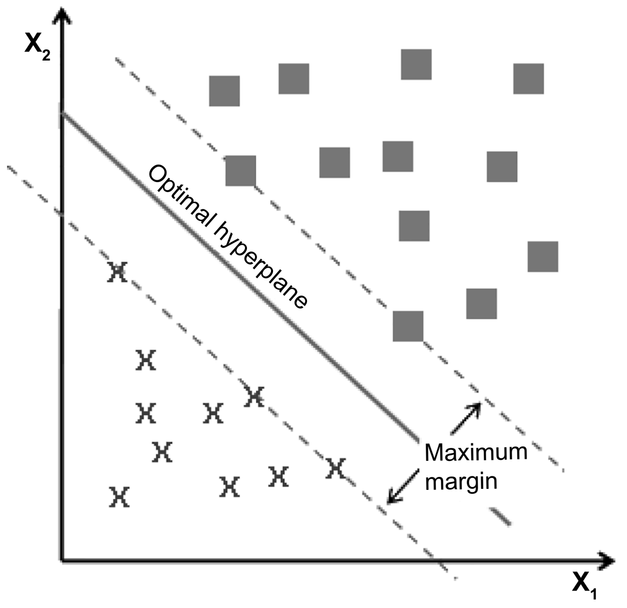

Figure 1. A binary dataset is composed of positive X_i^+ and negative X_i^- labeled values. For purposes of generalizing a dataset the hyperplane with the largest margin gives the best results, although there can be several hyperplanes that can optimally separate it.

$$w^* = \underset{w \in \Re^D}{\arg\min}\, F(w) := \frac{1}{2}\| w \|^2 + CR(w), \tag{1}$$

where w is a normal vector to the separating hyperplane, $\frac{1}{2}\| w \|^2$ is a quadratic regularization term and $C > 0$ is the fixed constant that scales the risk function.[10–13] Equation 1 is called the primal formulation.[14] By using Lagrange multipliers, the primal formulation can be presented in its dual form:

$$L(\alpha) = \arg\max\left\{ \sum_{i=1}^{n} \alpha_i - \sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j Y_i Y_j K(X_i, X_j) \right\} \tag{2}$$
$$\textit{subject to } 0 \le \alpha_i \le C \textit{ and } \sum_{i=1}^{n} \alpha_i Y_i = 0$$

where C is a fixed constant, $\left(X_i, Y_i\right)_{i=1}^{n}$ is a training set, α_i are Lagrange multipliers, $K(X_i, X_j)$ is the value of the kernel matrix defined by the inner product $\langle X_i, X_j\rangle$ (when a linear kernel K is used) and $Y_i \in \{\pm 1\}$ is a class label.[4]

The dual formulation has the same optimal values as the primal, but the main advantage of this representation is the use of the "kernel trick" (see Fig. 2). Since SVMs can only classify data in a linear, separable feature space, the role of the kernel function is to induce such feature space by implicitly mapping the training data into a higher dimensional space where data is linearly separable.[14,15] There are two main approaches for large-scale SVM training algorithms: those that solve the primal SVM formulation, shown in Equation 1, by a gradient-based method (primal estimated subgradient solver for SVM, careful quasi-Newton stochastic gradient descent, forward looking sub-gradient, etc.) and those that solve the dual formulation of Equation 2 by quadratic programming (QP) methods (SVM for multivariate performance measure, library for large linear classification and bundle method for risk minimization, etc.).[4,11,16,17] There are options that do not fall into these categories, such as the optimized cutting plane algorithm (OCA), which uses an improved cutting plane technique and is based on the work of SVM for multivariate performance measure (SVM$^{\text{perf}}$) and bundle method for risk minimization. OCA has fast convergence compared to methods like stochastic gradient descent and primal estimated sub-gradient solver for SVM (Pegasos), and it has shown good classification results and offers computational sublinear scaling.[13] Nevertheless, the use of a QP solver to solve a linear constraint problem (where each linear constraint is a cutting plane) makes it a complex approach to implement, even if the number of constraints is drastically lower than the data dimensionality. Gradient-based methods tend to be fast algorithms (especially those that use stochastic gradient descent) and have good generalization capabilities. However, they are highly dependent on step size to obtain a good speed of convergence. If the step size is not chosen carefully or it does not have an adjustment criteria, this can produce slow convergence.[4] The dual QP methods can handle kernels easily and can converge quickly by combining them with other optimization techniques. The main disadvantage of these methods is the computational complexity of the quadratic programming solvers and the fact that they are more difficult to implement than a gradient descent method or an EA.[4,18–21]

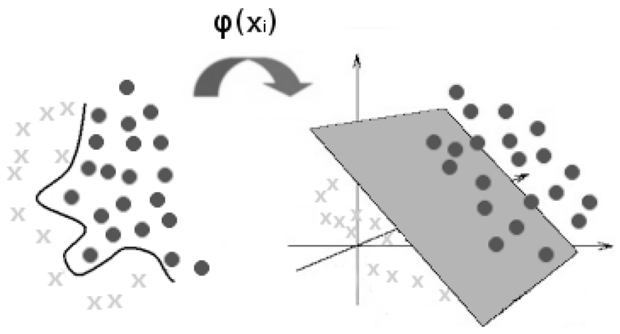

Figure 2. Datasets that are not linearly separable may be separated by a hyperplane in higher dimensions after applying the kernel trick.

In the past years, several evolutionary computation-based training algorithms for SVM have been proposed.[22–25] These algorithms solve the dual formulation (Equation 2), tend to be easy to implement and have shown good results for small amounts of data. The disadvantage on their implementation is their computational complexity of $O(n^2)$ or higher, where n represents the number of training samples. Since the complete kernel is needed on each iteration to calculate the fitness function, as the number of training samples grows, the time needed to process the data will increase drastically.

Evolutionary algorithms. EAs are global optimization methods that scale well to higher dimensional problems. They are robust with respect to noisy evaluation functions, and can be implemented and parallelized with relative ease.[26] Even when premature convergence to a local extremum may occur, it has been proven that an algorithm that is "not quite good" or "poor" at optimization can be excellent at generalization for a large-scale learning task.[4]

This work presents a series of parallelized algorithms based on the KA algorithm as fitness function combined with Artificial Bee Colony (ABC), micro-Artificial Bee Colony (μABC), Differential Evolution (DE) and Particle Swarm Optimization (PSO), in order to solve the SVM learning problem. The EA algorithms combined with KA were chosen based on good results shown in other areas, their exploration and exploitation capabilities, and low computational complexity.[27–33]

Large-scale training algorithms for SVMs using EA is a promising field that has not been well explored. Although parallelization is a highly desirable approach to the large-scale classification problem, most large-scale SVM training algorithms do not take this into consideration to obtain better results in a shorter amount of time. This is in part because testing complex parallel applications to guarantee a correct behavior is challenging; in scenarios, such as where inherent data dependencies exist, a complex task cannot be partitioned because of sequential constraints, making parallelization less convenient.[3,4] One of the main goals in parallelizing an EA is to reduce the search time. This is a very important aspect for some classes of problems with firm requirements on search time, such as in dynamic optimization problems and real-time planning.[34]

Protein structure prediction. A protein structure (PS) is the three-dimensional arrangement of atoms in a protein molecule.[35] These structures arise because particular sequences of amino-acids in polypeptide chains fold to generate, from linear chains, compact domains with specific 3D structures (Fig. 3). The folded domains can serve as modules for building up large assemblies such as virus particles or muscle fibers, or they can provide specific catalytic or binding sites, as found in enzymes or proteins that carry oxygen or regulate the function of DNA. PSP predicts the three-dimensional strucutres of a protein by using its first structure, its amino-acid sequence, to predict its folding and its secondary, tertiary and quaternary structure.[36,37] This makes PSP an essential tool in proteomics since the molecular function of a protein depends on its threedimensional structure, which is often unknown.

In the past 50 years there has been enormous growth in the available information regarding genomic sequences, to the point that the pace is difficult to follow. At present, more protein coding sequences are known than their three-dimensional structures. Protein folding is a large-scale problem because 20 different amino acids can generate such a large number of combinations, and there are also many ways for different amino-acid sequences to generate similar structural domains in proteins.[35] It has been suggested that many proteins contain enough information in their amino-acid sequences to determine their three-dimensional structure, making possible the prediction of new three-dimensional structures from an amino-acid sequence since it is known that sequence similarity does confer structural similarity.[38,39] Furthermore, to understand the biological function of proteins it is necessary to deduce or predict the three-dimensional structure from the amino-acid sequence, since their functional properties depend upon their structures. If the predictions are accurate enough, the gap between the growing amount of sequence information and their corresponding structures can be diminished.

PSP is, overall, an optimization problem where each amino acid can be characterized by several structural features. A good prediction of these features helps to obtain better models for the 3D-PSP problem. These features can be predicted as classification/regression problems, where the goal is to determine the shape (known as fold) that a given amino-acid sequence will adopt. The problem can take two possible directions.[40] The sequence may adopt a new fold, or bear resemblance to an existing fold in some protein structure database:

- If two sequences share evolutionary ancestry, they are called homologous and the structure for the query protein can be built by choosing the structure of the known homologous sequence as a template.

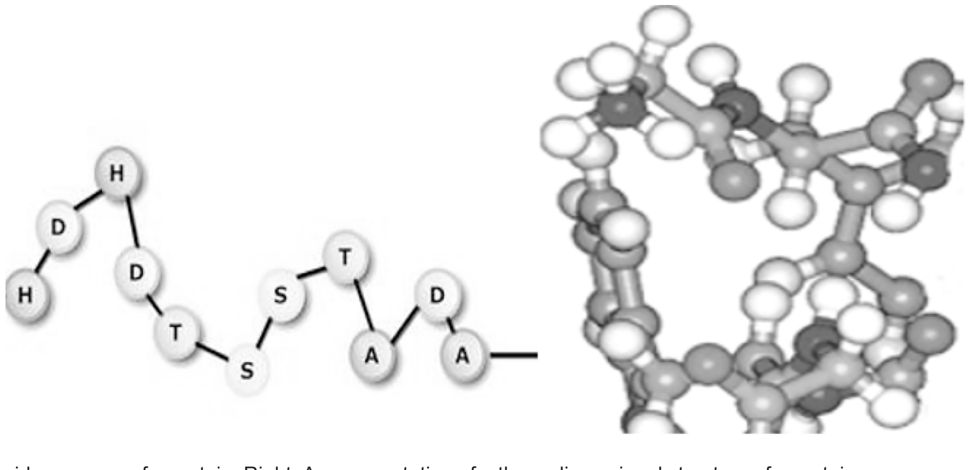

Figure 3. Left: Amino-acid sequence of a protein. Right: A representation of a three-dimensional structure of a protein.

- If no template structure is found for the query protein, the structure must be built from scratch.

Many methods have been developed to assign folds to a protein coding sequence.[41] These methods can be divided into three groups: sequencestructure homology recognition methods, threading methods and machine-learning-based methods. Sequence-structure homology and threading methods methods align the target sequence onto known structural templates and calculate their sequence-structure compatibilities, using, for example, environment-specific substitution tables or pseudo-energy-based functions to calculate if it is possible that a template is a fold of a target sequence.[42,43] Sequence-structure homology methods (like FUGUE and 3DPSSM) fail when two proteins are structurally similar, but share little in the way of sequence homology.[44,45] Threading methods (such as THREADER) depend on data derived from solved structures, but the number of proteins whose structure has been solved is much smaller than the number of proteins that have been sequenced.[46] Machine learning-based methods for protein fold recognition, like the approach presented in this paper, see the problem as a fold classification problem, where a classifier is built using a dataset with sequences of features of proteins with a known structure. The classifier can assign a structure-based label to an unknown protein (one that has not yet been solved).

In recent years, a number of diffierent SVM-based methods have been developed, producing better results than those obtained by pairwise sequence comparisons.[40,43,47–49] These algorithms have made improvements in the detection of homologous structures with low levels of sequence similarity (remote homology detection).

Most of the state of the art for PSP classification is not focused on large-scale data, and even if some approaches have shown good results in small-scale PSP classification, most use versions of SVM reliant on kernel functions or neural networks; these do not scale well as the dimension and/or the number of data to classify grows.[48,50–54] Because of this, some approaches tend to select an optimized feature subset with a moderate number of samples to improve the generalization performance of the SVM instead of using the complete dataset. This reduces the amount of data to compute, making it more practical to process with the original SVM approach, but also more time consuming since the dataset needs to be selectively preprocessed.[55] These methods might be good for small or medium amounts of data, but protein folding, because of its combinational nature, can generate an immense amount of data to process. This is where an algorithm especially designed for large-scale data is needed. Sequencing projects are fast at producing protein coding sequences, but only a small portion of protein coding sequences have experimentally solved 3D structures. This is due to the expensive and timeconsuming laboratory methods, such as X-ray crystallography and nuclear magnetic resonance (NMR).[41] This problem is becoming more pressing as the number of known protein coding sequences expands as a result of genome and other sequencing projects.[54] Because of this, tools that can predict PS rapidly and accurately, like the one presented in this paper, are needed. The full potential of genome projects will be realized only once we discover and understand the functions of these new proteins. This understanding will be facilitated by structural information for all or almost all proteins.

Methods

The kernel adatron algorithm. The Adaptive Perceptron algorithm (or Adatron) was first introduced by J. K. Anlauf and M. Biehl in 1989 for training linear classifiers.[56] This algorithm was proposed as a method for calculating the largest margin classifier. The Adatron is used for on line learning perceptrons and guarantees convergence to an optimal solution, when this exists.[57]

In 1998, T. Fries et al proposed the KA algorithm. Basically, the KA algorithm is an adaptation of the Adatron algorithm for classification with kernels in high-dimensional spaces.[5] It combines the simplicity of implementation of Adatron with an SVM's capability of working in nonlinear feature spaces to construct a large margin hyperplane using online learning.[15]

An advantage of KA algorithm is the use of gradient ascent instead of quadratic programming, which is easier to implement and significantly faster to calculate.

To implement KA algorithm, it is necessary to calculate the dot product $w \cdot X_i$, where X is the set of training points and w denotes the normal vector to the hyperplane that divides the classes with a maximum margin (Fig. 1). Since the kernel K is related to the high-dimensional mapping $\varphi(X_i)$ by equation

$$K\left(X_i, X_j\right) = \varphi\left(X_i\right) \cdot \varphi\left(X_j\right), \tag{3}$$

where the normal vector w to the separating hyperplane, can be expressed as

$$w = \sum_{i=1}^{n} \alpha_i Y_i \varphi\left(X_i\right), \tag{4}$$

then, by using the lineal kernel K, the dot product can be expressed as

$$z_i = \sum_{j=1}^{n} \alpha_j Y_j K\left(X_i, Y_j\right). \tag{5}$$

To update the multipliers, a change in α_i must be proposed to be evaluated. The change can be calculated as follows

$$\delta\alpha_i = \eta\left(1 - \gamma_i\right), \tag{6}$$

$$\gamma_i = Y_i z_i, \tag{7}$$

where η is the step size and $\delta\alpha_i$ is the proposed change to α_i. If $\alpha_i + \delta\alpha_i \leq 0$ it would result in a negative α_i. To avoid this problem, α_i is set to 0. Otherwise, update $\alpha_i \leftarrow \alpha_i + \delta\alpha_i$. The bias b (Fig. 1) can be obtained as follows:

$$b = \frac{1}{2}\left(min\left(z_i^+ \right) + max\left(z_i^- \right) \right), \quad (8)$$

where z_i^+ are the patterns with class label +1 and z_i^- are those with class label –1.

The pseudocode is described briefly in Algorithm 1.

Algorithm 1. Kernel Adatron Algorithm.

```
 1: Initialize $\alpha_i = 1$.
 2: repeat
 3:     For $(X_i, Y_i)$ calculate $z_i$ with Equation 5.
 4:     Calculate $\gamma_i$ with Equation 7.
 5:     Calculate $\delta\alpha_i$ with Equation 6.
 6:     if $(\alpha_i + \delta\alpha_i) \leq 0$ then
 7:         $\alpha_i = 0$
 8:     end if
 9:     if $(\alpha_i + \delta\alpha_i) > 0$ then
10:         $\alpha_i = \alpha_i + \delta\alpha_i$
11:     end if
12:     Calculate $b$ with Equation 8
13: until The stopping criteria is met.
```

Evolutionary algorithms. Evolutionary computing is a subfield of artificial intelligence that includes a range of problem-solving techniques based on principles of biological evolution. The principles for using evolutive processes to solve optimization problems originated in the 1950s.[58]

The EA are optimization methods that are part of evolutionary computing, applying models based on biological evolution. In EA, a population of possible solutions is composed of individuals that can be compared according to their aptitude to improve the population; the most qualified candidates are those that obtain better results by a fitness function evaluation. The evolution of the population is obtained through iterations, in which a series of operations are applied to the individuals of the population (reproduction, mutation, recombination or selection), from these operations a new set of potentially better solutions are generated. The way the population evolves the possible solutions, and the way it chooses the new global best solutions, is something inherent to each EA.[59]

A swarm intelligence algorithm is based on swarms that occur in nature; PSO and ABC are two prominent swarm algorithms. There is a debate on whether swarm intelligence-based algorithms are EAs or not, but since one of the inventors of PSO refers to it as an EA, and swarm intelligence algorithms are executed in the same general way as EAs, by evolving a population of candidate problem solutions that improves with each iteration, we consider swarm intelligence to be an EA.[59,60]

As mentioned before, the KA algorithm requires the α_i value to be adjusted through iterations. In this approach, the adjustment is made using EA (Fig. 4). This type of algorithm was chosen as an optimization method because they are easy to implement, to parallelize and have shown good results in diverse areas such as computer vision, image processing and path planning.[27,28,30,61–63]

Artificial bee colony algorithm. The ABC algorithm was first introduced by Karaboga in 2005.[64] This algorithm is based on honey bee foraging behavior. The bees are divided into three classes:

- Employed: Bee with a food source.
- Onlookers: Bee that watches the dances of employed bees and choose food sources depending on dances.
- Scouts: Employed bee that abandons its food source to find a new one.

Each food source is equivalent to a possible solution to the optimization problem and, as in nature, individuals are more likely to be attracted to sources with a larger amount of food (a better result obtained by the fitness function). For each food source, only one employed bee is assigned, and when it abandons its food source it becomes a scout. The number of the onlooker bees is also equal to the number of solutions in the population.

Initially, ABC algorithm generates a random population P of n solutions. Each solution $x_i \in P$ is a D-dimensional vector, to be evaluated by a fitness function $f()$, also known as food source. The algorithm searches iteratively for the better food sources based on the findings made by employed, onlooker and scout bees. First, the i-th employed bee generates a random modification in the j-th position of its corresponding food source x_{ij}, producing a new potential food source v_i. The potential food source can be obtained by Equation 9

$$v_{ij} = x_{ij} + \phi_{ij}(x_{ij} - x_{kj}), \quad (9)$$

where $k \in 1, 2, \ldots, n$ is a randomly chosen index diffierent from i and ϕ_{ij} is a uniformly distributed random number between [–1, 1].

If the amount of nectar (the value obtained by the fitness function) is greater than the old one, the employed bee takes it as its new food source x_i. Otherwise, the food source x_i remains unchanged.

Once positions of the employed bees have been updated, the information is shared with the onlooker bees. Onlooker bees choose their food sources based on a probability p_i that is directly related to the amount of nectar. The value of p_i is obtained as follows

$$p_i = \frac{f_i}{\sum_{m=1}^{n} f_m}, \quad (10)$$

where f_i is the fitness value of the i-th food source. p_i is choosen by a roulette wheel selection mechanism (the better the i-th solution, the higher its chances of being selected).

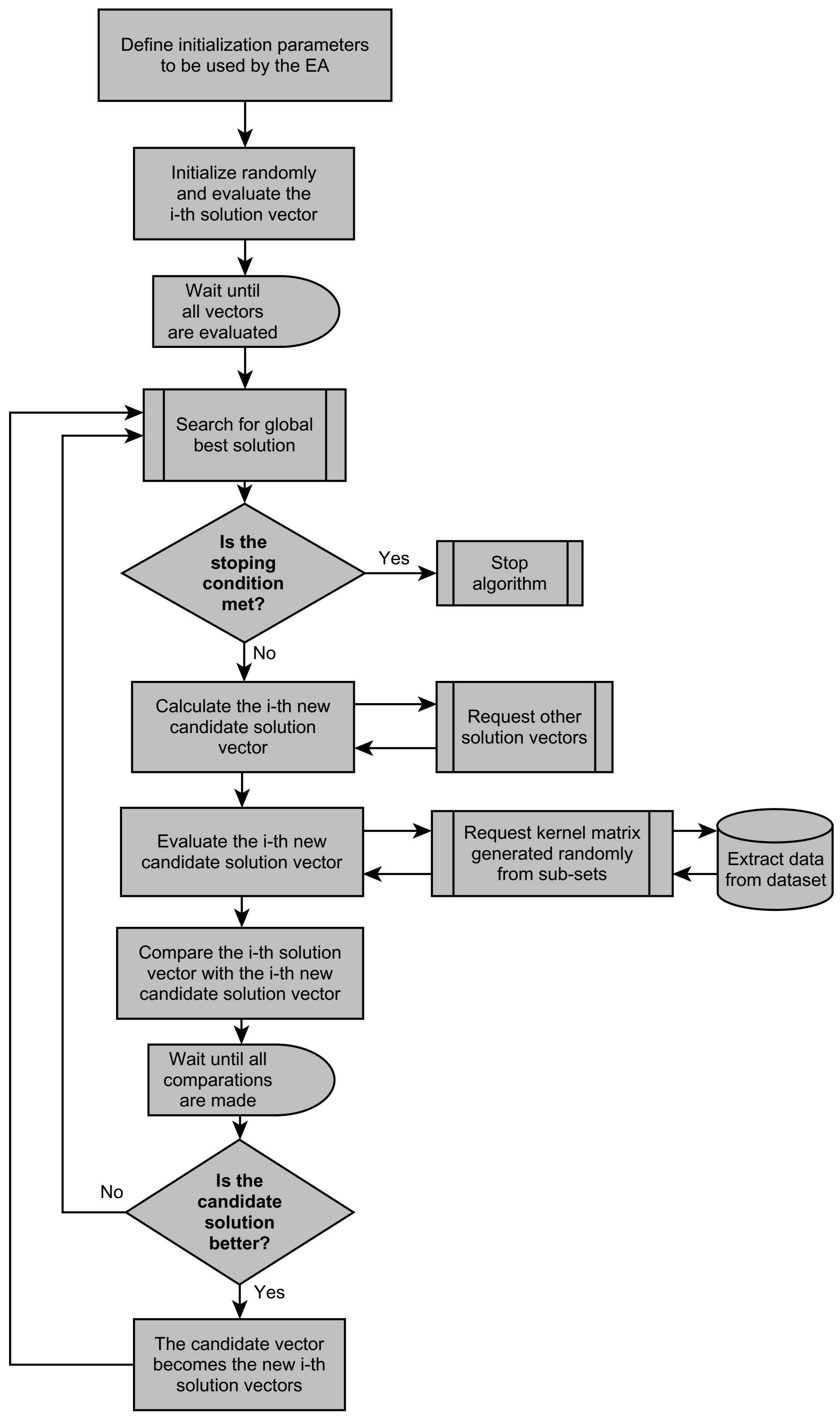
Define initialization parameters to be used by the EA
Initialize randomly and evaluate the i-th solution vector
Wait until all vectors are evaluated
Search for global best solution
Is the stoping condition met?
Yes
Stop algorithm
No
Calculate the i-th new candidate solution vector
Request other solution vectors
Evaluate the i-th new candidate solution vector
Request kernel matrix generated randomly from sub-sets
Extract data from dataset
Compare the i-th solution vector with the i-th new candidate solution vector
Wait until all comparations are made
No
Is the candidate solution better?
Yes
The candidate vector becomes the new i-th solution vectors

Figure 4. The diagram explains the basic idea behind the algorithm described in this paper.

A new potential food source v_i is calculated using Equation 9, where x_{ij} is selected based on the roulette wheel selection result. And, as with employed bees, if the amount of nectar improves, v_i replaces x_i; otherwise, x_i remains unchanged.

If a position x_i cannot be improved through a certain number of iterations, the i-th food source is abandoned. If this occurs, the scout bee changes its actual food source for a new food source to replace x_i as follows

$$x_{ij} = lb_j + rand(0,1)(ub_j - lb_j), \tag{11}$$

where $rand(0, 1)$ is a normally distributed random number within [0, 1], and lb and ub are lower and upper bounds of the j-th dimension, respectively. The pseudocode is briefly described in Algorithm 2.

μArtificial bee colony algorithm. The μABC algorithm was first introduced by Rajasekhar in 2012.[29] This algorithm is a variant of the ABC algorithm with a small population (only 3 bees).

The population of bees evolves through iterations and only the best bee is kept unaltered, whereas the rest of the bees are reinitialized with modifications based on the food source with the best fitness.

After the employed and onlooker phases have been completed (in the same way as in the ABC algorithm) the population is ranked according to its fitness values. The bee with the best fitness remains in its food source, while the second best fitness is moved to a position near to the best one in order to facilitate a local search. The bee with the worst position is initialized to a random position to avoid premature convergence.

Unlike ABC, more than one variable is modified from the food source. For each parameter x_{ij}, a uniformly distributed random number $rand_{ij}(0, 1)$ is generated and if this number is less than the Frequency Control Rate (FCR) parameter, which is user defined, then the variable x_{ij} is modified as follows

$$v_{ij} = \begin{cases} x_{ij} + \phi_{ij}(x_{ij} - x_{kj}), & if\ rand(0,1) \leq FCR \\ x_{ij} & \text{otherwise} \end{cases} \tag{12}$$

Algorithm 2. Artificial Bee Colony Algorithm.

```
 1: Initialize $x_i$
 2: repeat
 3:     Produce a new solution $v_i$ for the employed phase
    with Equation 9.
 4:     if $f(v_i) < f(x_i)$ then
 5:        $x_i \leftarrow v_i$.
 6:     end if
 7:     Calculate the probability values $p_i$ with Equation 10
    for the solution $x_i$.
 8:     Produce a new solution $v_i$ for the onlooker phase
    with Equation 9, selecting $x_i$ based on $p_i$.
 9:     if $f(u_i) < f(x_i)$ then
10:        $x_i \leftarrow v_i$.
11:     end if
12:     if $x_i$ is an abandoned solution for the scout phase then
13:        Replace $x_i$ by using the Equation 11.
14:     end if
15: until The stopping condition is met.
```

The value of ϕ_{ij} is a uniformly distributed random number, maintained in the range of $[-RF, RF]$, where RF is the range factor. RF changes automatically during the search by tuning its value in accordance with Rechenberg's 1/5 rule. This rule states that 1/5 of the total mutations in every t iterations $\varphi(t)$ should be successful mutations. According to the number of successes $\varphi(t)$, the value of RF is adjusted according to

$$RF_{new}(it+1) = \begin{cases} RF_{old}(t) * 0.85 & if\ \varphi(t) < 1/5 \\ RF_{old}(t)/0.85 & if\ \varphi(t) > 1/5 \\ RF_{old}(t) & if\ \varphi(t) = 1/5 \end{cases} \tag{13}$$

The pseudocode is briefly described in Algorithm 3.

Algorithm 3. Micro Artificial Bee Colony Algorithm.

```
 1: Initialize $x_i$
 2: repeat
 3:     Produce a new solution $v_i$ for the employed phase with
        Equation 12.
 4:     if $f(v_i) < f(x_i)$ then
 5:        $x_i \leftarrow v_i$.
 6:     end if
 7:     Calculate probability values $p_i$ with Equation 10 for solution $x_i$.
 8:     Produce a new solution $v_i$ for the onlooker phase with
    Equation 12, selecting xi based on $p_i$.
 9:     if $f(u_i) < f(x_i)$ then
10:        $x_i \leftarrow v_i$.
11:     end if
12:     Move second best solution $x_{2b}$ to a position very close to
    best solution $x_{1b}$.
13:     Move worst solution $x_{3b}$ to a random position.
14: until The stopping condition is met.
```

Differential evolution. DE was first introduced by R. Storn and K. V. Price in 1995.[65] In DE each individual x_i of the population is a D-dimensional vector that represents a candidate solution from a set of n solutions. Each individual, called a vector, is evaluated by a fitness function $f()$ to define its strength as a solution. The fundamental idea behind DE is creating new candidate solutions based on other solutions that have been previously found. DE takes the difference vector between two randomly chosen individuals, x_{r2} and x_{r3}, and adds a scaled version of this vector to a third individual, chosen randomly x_{r1} or the best individual x_b in the population. For the algorithm described in this paper, we used $x_{r1} = x_b$. This new individual is called a mutant vector v_i

$$v_i = x_{r1} + F(x_{r2} - x_{r3}), \tag{14}$$

where F is a user-defined scaling factor. This mutant vector v_i is later combined with x_i by crossover to create a candidate solution to be evaluated by an objective function. The crossover is implemented as follows

$$u_{ij} = \begin{cases} v_{ij} & if\ (r_{ij} < CROV)\ or\ (j = J_r) \\ x_{ij} & otherwise \end{cases} \quad for\ i = 1,2,\ldots,n; \quad and\ j = 1,2,\ldots,D; \tag{15}$$

where u_{ij} is the crossed vector, r_{ij} is a random number between [0, 1], $CROV$ is the user-defined constant crossover rate $\in [0, 1]$ and J_r is a random integer $\in [0, D]$ redefined on each iteration. The pseudocode is briefly described in Algorithm 4.

Algorithm 4. Differential Evolution Algorithm.

```
1: Initialize $F = [0.4, 0.9]$, $CROV$ and $x_i$
2: repeat
3:     For each $x_i$ choose three random integers ($r1$, $r2$, $r3$), where
   $r1 \neq r2 \neq r3$ and $r1$, $r2$, $r3 \in [1, n]$.
4:     Generate $n$ mutant vectors with Equation 14.
5:     Generate $n$ crossed vectors with Equation 15.
6:     if $f(u_i) < f(x_i)$ then
7:        $x_i \leftarrow u_i$.
8:     end if
9: until The stopping condition is met.
```

Particle swarm optimization. The PSO algorithm was first introduced by Kennedy and Russell in 1995.[66] This algorithm exploits a population of potential solutions. The population of solutions is called a swarm and each individual from a swarm is called a particle. A swarm is defined as a set of n particles. Each particle i is represented as a D-dimensional position vector x_i, which is evaluated by a fitness function $f()$. Based on the results of the evaluation, it is easy to measure improvement in new particles compared to old ones. The particles are assumed to move within the search space iteratively. This is done by adjusting their position using a proper position shift, called velocity v_i. For each iteration t, the velocity changes by applying Equation 16 to each particle.

$$v_i(t+1) = \omega v_i(t) + c_1\varphi_1(P_{ibest} - x_i) + c_2\varphi_2(P_{gbest} - x_i) \qquad (16)$$

where φ_1 and φ_2 are random variables uniformly distributed within [0,1]; c_1 and c_2 are weighting factors, also called the cognitive and social parameters, respectively; ω is called the inertia weight, which decreases linearly from ω_{start} to ω_{end} during iterations. P_{ibest} and P_{gbest} represent the best position visited by a particle and the best position visited by the swarm before the current iteration t, respectively.

The position update is applied by Equation 17 based on the new velocity and the current position.

$$x_i(t+1) = x_i(t) + v_i(t+1). \qquad (17)$$

The basic algorithm is as follows:

Algorithm 5. Particle Swarm Optimization.

```
1:  Initialize $c_1$, $c_2$, $v_i$ and $x_i$
2:  $P_{ibest} \leftarrow x_i$.
3:  Select from $x_i$, $P_{gbest}$.
4:  repeat
5:      Obtain velocity $v_i$ with Equation 16.
6:      Update position $x_i$ with Equation 17.
7:      if $f(x_i) < f(P_{ibest})$ then
8:          $P_{ibest} \leftarrow x_i$
9:          if $f(P_{ibest}) < f(P_{gbest})$ then
10:             $P_{gbest} \leftarrow P_{ibest}$
11:         end if
12:     end if
13: until The stopping condition is met.
```

To solve the uncontrolled increase of magnitude of the velocities (swarm explosion effect), it is often necessary to restrict the velocity with a clamping at desirable levels, preventing particles from taking extremely large steps from their current positions.[67]

$$v_{ij}(t+1) = \begin{cases} v_{max} & if\ v_{ij}(t+1) > v_{max}, \\ -v_{max} & if\ v_{ij}(t+1) > -v_{max} \end{cases}$$

Although the use of a maximum velocity threshold improves the performance, by controlling the swarm explosions, without the inertia weight the swarm would not be able to concentrate its particles around the most promising solutions in the last phase of the optimization procedures.[67]

Kernel adatron trained with evolutionary algorithms. The basic idea behind the proposed algorithms is to use a "divide and conquer" strategy, where each individual in the population of the EA (vector in DE, particle in PSO, food source in ABC and μABC) is seen as a sub-process, in this case a thread (Fig. 5), that will solve a part of the whole problem. Once each sub-process reaches a result, it is compared to the results of its peers to improve future results.

DE, PSO, ABC and μABC are easily parallelized because each individual can be evaluated independently. The only phases in which the algorithms require communication between their individuals are the phases that involve mutation and the selection of the fittest individual. Also, the process to obtain the kernel matrix can be easily parallelized by dividing the process into several subtasks. For this approach, a lineal kernel is used (represented by the dot product $\langle X_i, X_j \rangle$), since it was the kernel that gave the best results.

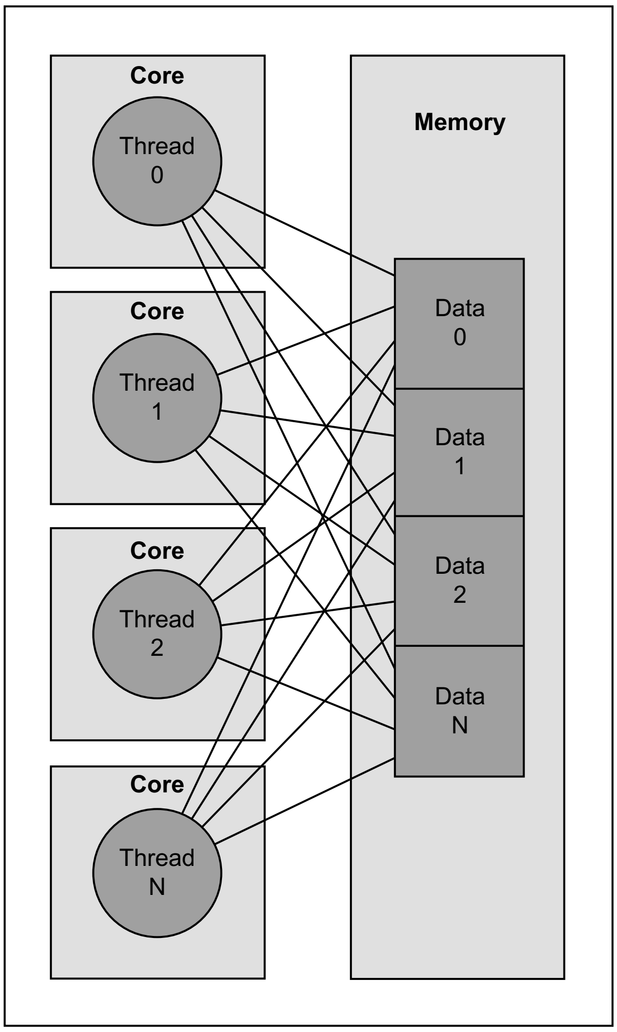

Figure 5. A thread is a component of a process. Multiple threads can exist within the same process; they are executed concurrently and share resources, such as memory.

On each variant of the proposed algorithm, individual x_i (particle, vector or bee) represents a D-dimensional vector composed of multipliers to be optimized over iterations by the EA. The fitness function $f()$ to be used by the EA is described by Equation 18:

$$f(x) = abs(1-\Theta) \tag{18}$$

where Θ is the margin between classes of the hyperplane, which can be estimated as follows:

$$\Theta = \frac{1}{2}(min(z_i^+) - max(z_i^-)). \tag{19}$$

The value z_i can be obtained with Equation 5. The values of z can be divided into z_i^+ and z_i^- depending on their class label, +1 and −1, respectively. The *KA* algorithm has the implementational simplicity of the Adatron model and can find a solution very rapidly compared to traditional methods like kernel-perceptron and SVM.[5] The algorithm comes with all the theoretical guarantees given by support vector theory for large margin classifiers, as well as the convergence properties studied in the statistical learning literature.[68] However, the algorithm uses basic operations and has a complexity of $O(n^2)$. Because of this, the algorithm has been modified so it can be trained using an EA with a computationally more attractive fitness function.

The main problem of KA is calculating the z_i values. This results in an impractical fitness function, since it turns the linear computational complexity of the EA into quadratic. To solve this problem, it is proposed to use subsets of values to approximate a subset of z_i for evaluating a candidate solution, instead of calculating each exact value of z_i. Each subset is generated randomly and uses a much smaller fixed number of values (defined as nvals in Algorithm 6) than the number of values contained by the kernel matrix. The fitness function is described in Algorithm 6.

Algorithm 6. Fitness Function.

```
1:  Initialize $n_{vals}$, $z^+_{min} = INFINITY$, $z^+_{max} = -INFINITY$
2:  Generate a vector $rvec$ with $n_{vals}$ number of integer elements.
    Where $rvec_i \in [0, n_{ts}]$
3:  for each element in $rvec_i$ do
4:      $z_i = \sum_{j=1}^{n_{vals}} \alpha_{rvec_j} Y_{rvec_j} K(X_{rvec_j}, X_{rvec_i})$
5:      if $z_i \in z^+$ and $z_i < z^+_{min}$ then
6:          $z^+_{min} = z_i$
7:      else
8:          $z^-_{max} = z_i$
9:      end if
10: end for
11: $\Theta = \frac{1}{2}(z^+_{min} - z^-_{max})$
12: return $abs(1 - \Theta)$
```

The number of data to be used by the fitness function nvals in this approach needn't necessarily increase drastically with an increase in the number of training samples of the data set n_{ts} or dimensionality of the problem. The value for n_{vals} was obtained from several tests done by running PSO on each variant of the algorithm on several datasets, and obtaining the average of the optimal number of samples needed by each approach. The value for n_{vals} was merely 400 data samples, which gave the best results in the tests made on the datasets mentioned in Results and Discussion Section. Since all the results were near 400 samples selected randomly, this number was taken as a constant number of samples for n_{vals} in all the tests, independently of n_{ts}.

The fitness function complexity is $O(1)$, if the kernel matrix K is previously computed, or $O(d)$ for any $K\left(X_{rvec_j}, X_{rvec_i}\right)$ value that is calculated by the fitness function, where d is the maximum number of non-zero features in any of the training samples.

Interdisciplinary computing and complex biosystems protein structure prediction benchmarks repository. The Interdisciplinary Computing and Complex BioSystems Protein Structure Prediction (ICOS PSP) benchmarks repository[I] contains datasets suitable for testing classification algorithms based on real data.[69,70] The dataset is based on PSP, aiming to predict the three-dimensional structures of amino-acid chains based on several structural features. The features are extracted by using a window of size Ω on amino-acid chain to predict the Coordination Number (CN) for residue i by using the information of its neighbors. Where a residue i refers to a specific amino-acid within the polymeric chain of a protein, the CN is the number of residues from the same protein that are in contact with a given residue in the native state. Two residues are said to be in contact when the distance between them is below a certain threshold. The dataset is derived from a set of 1050 protein chains and approximately 260,000 amino-acids (instances) selected using the PDB-REPRDB database. In order to predict the real-valued CN using classification techniques, the continuous domain was mapped onto a finite set of categories.[II] Two different criteria were used to generate sets with two, three and five classes (or states) to form classes with balanced and imbalanced class distribution, uniform frequency and uniform length, respectively.[71,72]

Binning is the simplest method to discretize a continuous-valued attribute by creating a specified number of bins. The bins can be created by uniform frequency or length. In both, arity k is used to determine the number of bins, which are associated with a distinct discrete value. For uniform length, the continuous range of a feature is evenly divided into intervals that have equal length and each interval represents a bin. In uniform frequency, an equal number of continuous

[I]The ICOS PSP benchmarks repository is available at http://ico2 s.org/datasets/psp_benchmark.html.

[II]The description of the dataset is available at http://ico2 s.org/datasets/psp/motivation.html.

values are placed in each bin.[72] For this dataset the bins are computed separately for each training set using all of its instances, and afterwards applied also to the corresponding test set. To construct the datasets, a Ω window size ranging from 0 to 9 amino acids was used. The primary sequence of the protein and the CN definition of each amino acid were extracted from the PDB file. As in,[73] a standard bootstrapping technique was used, which is useful for the robust estimation of prediction accuracy and its error; that is, a dataset of 1050 protein chains was randomly divided into 2 groups: the training set of 950 chains and the test set of 100 chains. This division of the whole dataset was repeated 10 times, resulting in 10 pairs of training and test sets. Each training set contains more than $2x10^5$ residues. For this paper, only the subset divided into two states was used since the approach is proposed for binary classification.

Results and Discussion

The data to classify was taken from the Interdisciplinary Computing and Complex BioSystems Protein Structure Prediction Benchmarks Repository and seven other datasets from diverse fields that are commonly used to test large-scale classifiers; the datasets are briefly described in Tables 1 and 2.

From the PSP dataset, only the subsets discretized with uniform length and uniform frequency, with window sizes ranging from 7 to 9, were used for training and generalization because of their density and dimensionality. The Astro-Ph dataset is focused on classifying abstracts of scientific papers from Physics ArXiv.[74] The Aut-Avn and Real-Sim classification datasets come from a collection of UseNet articles from four discussion groups: for simulated auto racing, simulated aviation, real autos and real aviation. CCAT and C11 are obtained from the Reuters RCV1 collection, and address the problem of separating corporate related articles.[3] The Worm dataset focuses on classifying worm RNA splices.[III,13]

The experiments were performed on an Intel® Core i7–3770™[IV] machine with 16 GB of RAM and Fedora Linux 20[V] operating system. The code was written in C++ using POSIX Threads[VI] and Armadillo.[75] For the implementation of the algorithms, the Armadillo random number generator was used; the C++ random number generator was more expensive computationally speaking and increased the execution time drastically.

For the experiments done in this section, our approach is compared against algorithms like OCA, SVM*light*, SVM*perf* and the original KA algorithm, from which the first three algorithms are large-scale SVM classifiers used in diverse fields.[1,2]

Something to be taken into account is that it is much easier to implement and parallelize EA algorithms than to implement or parallelize the QP solvers used by OCA, SVM*light* and SVM*perf*.[13,74,76] The work presented in this paper was developed and tested on a multi-core computer, but since the algorithm is easily parallelizable, it can be implemented to run on a computer cluster with fewer complications than implementing a parallelized version of the previously mentioned algorithms for the same cluster. It is expected that, by using this type of hardware, the training and evaluation time can be reduced, even when processing a considerably larger amount of data.

For the EA fitness function, a linear kernel was used in all the algorithms since it gave the best results in the generalization tests. Several tests were made using a radial basis function kernel. In general, the results showed a slight increase in the training accuracy (not sufficient to compete with the other approaches in the training phase), the generalization accuracy decreased slightly and the processing time increased because of the extra operations that had to be performed to calculate the kernel. Because of this, only the results obtained with the linear kernel are shown.

Previous to the tests, from each dataset a subset of 4000 training samples was randomly extracted and normalized for binary training classification and cross-validation. Because of hardware limitations, the amount of training samples used on each dataset is not large-scale, so that it could be stored in the computer's memory. However, since the KA algorithm possess the guarantees given by the support vector theory and, as explained later in this section, the algorithm scales well

Table 1. Brief description of large-scale datasets. Density denotes the average percentage of non-zero features of the data vectors.

DATASET	DIMENSION	DENSITY
Astro-Ph	99757	0.08%
Aut-Avn	20707	0.23%
C11	47236	0.16%
CCAT	47236	0.16%
RCV1	47236	0.18%
Real-Sim	20958	0.23%
Worm	804	25.00%

Table 2. Brief description of the ICOS PSP dataset.

UNIFORM:	Ω	DIMENSION	DENSITY
Length	7	300	86.04%
	8	340	86.98%
	9	380	88.79%
Frequency	7	300	87.24%
	8	340	87.07%
	9	380	89.17%

[III]The datasets can be obtained at http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html and http://users.cecs.anu.edu.au/~xzhang/data/

[IV]Intel and Intel Core are trademarks of Intel Corporation.

[V]Fedora is a trademark of Red Hat, Inc.

[VI]For more information about POSIX: http://pubs.opengroup.org/onlinepubs/9699919799/

with the increase in the amount of data and dimensionality, the algorithm can easily be used with a larger amount of data without problems.[68] The dimensionality and density of the datasets can be seen in Tables 1 and 2. The generalization accuracy was obtained by applying a 10-fold cross-validation to each dataset. To test the accuracy of training capability of each algorithm, the SVM was trained using 3600 training samples per run, which represents the n_{ts} value for a dataset, and 400 samples were used for testing.

The values used to train the SVM with each EA were obtained by running PSO on each variant of the algorithm to determinate the optimal values. This is not to be confused with the PSO variant that uses KA to classify data. The following values were used by the EAs while using the large-scale datasets:

- The μABC version used: $RF = 0.0001$, $C = 0.0001$, $FCR = 0.0001$ and maximum of 5 attempts before abandoning a food source.
- The ABC version used: $C = 2$, ϕ_{ij} values ranging between [–2, 2], 5 food sources and a maximum of 9 attempts before abandoning a food source.
- The DE algorithm used: $C = 2.38958$, $F = 1.87016$ and $CROV = 0.9$ and 6 vectors.
- The PSO algorithm used: $v_{max} = 1.49684$, $w_{start} = 1.18472$, $w_{end} = 0.000511895$, $c_1 = 1.03971$ $c_2 = 1.48063$, $C = 6.74659$ and 15 particles.

For the PSP dataset, the following values were, used by the EAs:

- The μABC version used: $RF = 0.001$, $C = 0.0001$, $FCR = 0.001$, with maximum of 5 attempts before abandoning a food source and a maximum of 25 iterations as stopping condition.
- The ABC version used: $C = 5$, ϕ_{ij} values ranging between [–2, 2], 8 food sources, a maximum of 9 attempts before abandoning a food source and a maximum of 20 iterations as stopping condition.
- The DE algorithm used: $C = 2.65435$, $F = 0.719909$ and $CROV = 0.1$, with 6 vectors and a maximum of 23 iterations as stopping condition.
- The PSO algorithm used: $v_{max} = 0.1$, $w_{start} = 0.0494229$, $w_{end} = 0.0001$, $c_1 = 1.13755$ $c_2 = 0.11384$, $C = 3.5$ with 10 particles and a maximum of 30 iterations as stopping condition.

The C value in SVM has two main purposes: it functions as constant that scales the risk function for the primal formulation in Equation 1 and it limits the values that any α_i can take in the dual formulation in Equation 2. In this paper, the value of C is used in the same way as in the dual formulation, for limiting the values of α_i. A total of 200 iterations was used as stopping condition by the EA for the datasets described in Table 1, because all the algorithms trained with PSO returned values close to 200 iterations as the optimal value for the stopping condition, with 200 being the highest number of iterations.

As stated in Section The Kernel Adatron Algorithm, the KA algorithm has appealing advantages such as the simplicity of implementation of Adatron and the capability of working in high-dimension feature spaces to construct a large margin hyperplane. But the main concern of implementing the original KA approach is working with the kernel matrix, since its computational complexity is of $O(d^{*}n_{ts}^{2})$, where d is the maximum number of non-zero features in any data vector of the training subset and n_{ts} is the number of training samples. Nevertheless, there are scenarios, such as that presented in Table 1, where the density of the data samples is low in most cases, so the number of operations to calculate the kernel matrix can be drastically reduced. On the other hand, independently of the density, if it is treated as a divide and conquer problem the computational complexity is reduced, at worst case scenario, to $O(d^{*}n_{ts}^{2}/t)$, where t is the number of threads. Methods like Sequential Minimal Optimization or chunking can be used to reduce the computational complexity, but these algorithms, in the worst case scenario, scale to $O\left(n_{ts}^{2}\right)$ and $O\left(n_{ts}^{3}\right)$, respectively, which makes them expensive computationally speaking.[77]

The approach proposed in this paper always uses, per iteration, a subset of randomly chosen training samples with a much smaller fixed size, and it is independent of the number of training samples n_{ts} in the dataset. Because of this, the complexity remains linear $O(d)$ ($O(d/t)$ if it is parallelized) even if the dataset increases in size. For all the experiments made using the datasets described in Table 1, a total of 60 randomly chosen training samples from a dataset were used every time the fitness function was called. For the PSP dataset, the number of samples used per fitness function call was 400, over three times more data than with the other datasets, but still a considerably small amount of samples considering the density and the complete number of samples. These values were also obtained with PSO. Since the approaches shown in this paper works with data subsets, some precision in the accuracy of the training phase is lost to gain a better generalization capability in a small amount of time.

For the approach shown in this paper, the EAs the computational complexity is linear $O(n)$, where n is the number of individuals in the population of the EA, and $O(d)$ for the fitness function, so the whole complexity of the algorithm is $O(n * b)$ (Table 3). Compared to SVM^{light} and *KA*, in which computational complexity is equal to higher than $O(d * n_{ts}^{2})$, the approach shown in this paper is more appealing.[4] Algorithms such as OCA and SVM^{perf} show a computational complexity of $O(d * n_{ts})$, which makes this approach competitive by comparison.[4,74]

As shown in Tables 7 to 18, our approach gave results in generalization and time tests (measured in seconds) that are competitive with or better than those shown by OCA, SVM^{light} and SVM^{perf}, though the accuracy in the training

Table 3. Computational complexity of the algorithms.

ALGORITHM	COMPLEXITY
KA	$O(d * n_{ts}^2)$
SVM^{light}	$O(d * n_{ts}^2)$
OCA	$O(d * n_{ts})$
SVM^{perf}	$O(d * n_{ts})$
EA approaches	$O(n * b)$

Table 4. Results obtained from the Friedman test were the sum of squares (SS), mean squares (MS), degrees of freedom (df), χ^2 value and *P* -value.

(**A**) Friedman test made to the Astro-Ph, Aut-Avn, C11, CCAT, RCV1, Real-Sim and Worm datasets.

SOURCE	SS	DF	MS	χ^2	*P*-VALUE
Columns	9.2857	3	3.0952	5.9091	0.1161
Error	23.7143	18	1.3175		
Total	33	27			

(**B**) Friedman test made to the PSP dataset.

SOURCE	SS	DF	MS	χ^2	*P*-VALUE
Columns	22.0833	3	7.3611	13.4746	0.0037
Error	7.4167	15	0.4944		
Total	29.5000	23			

Table 5. Mean rank obtained from the Friedman test for each solver.

(**A**) Mean rank from the Astro-Ph, Aut-Avn, C11, CCAT, RCV1, Real-Sim and Worm datasets.

	DE	*PSO*	SVM^{light}	SVM^{perf}
Mean	2.2857	1.8571	3.4286	2.4286

(**B**) Mean rank from the PSP dataset.

	DE	*PSO*	SVM^{light}	SVM^{perf}
Mean	3.2 500	3.4167	2.3333	1

Table 6. Results from the Astro-Ph dataset. The best global results are underlined and the best results obtained by our approach are written in bold letters.

ALGORITHM	TRAINING	GENERALIZATION	TRAINING TIME
μABC	94.50%	92.65%	0.0243
ABC	**94.58%**	93.63%	0.0650
DE	94.56%	**93.80%**	**0.0191**
PSO	94.53%	93.77%	0.0212
KA	94.61%	92.68%	12.0500
SVM^{light}	99.27%	95.33%	0.2430
SVM^{perf}	95.82%	93.85%	0.0195
OCA	100.00%	93.25%	0.0282

Table 7. Results from the Aut-Avn dataset.

ALGORITHM	TRAINING	GENERALIZATION	TRAINING TIME
μABC	97.23%	94.98%	0.0216
ABC	97.20%	94.58%	0.0725
DE	97.28%	96.13%	**0.0172**
PSO	**97.65%**	**96.95%**	0.0198
KA	97.34%	94.95%	12.5613
SVM^{light}	99.70%	95.65%	0.1380
SVM^{perf}	98.52%	96.03%	0.0102
OCA	100.00%	90.10%	0.0384

Table 8. Results from the C11 dataset.

ALGORITHM	TRAINING	GENERALIZATION	TRAINING TIME
μABC	85.42%	81.33%	0.0221
ABC	86.52%	86.85%	0.0129
DE	**87.01%**	**87.58%**	**0.0121**
PSO	86.55%	86.44%	0.0198
KA	87.92%	83.80%	11.5700
SVM^{light}	98.12%	87.58%	0.0111
SVM^{perf}	98.55%	87.58%	0.0102
OCA	100.00%	72.84%	0.0479

Table 9. Results from the CCAT dataset.

ALGORITHM	TRAINING	GENERALIZATION	TRAINING TIME
μABC	90.49%	86.18%	0.0287
ABC	91.11%	86.78%	0.0436
DE	**91.82%**	**86.98%**	**0.0187**
PSO	91.74%	86.55%	0.0387
KA	90.75%	86.58%	12.5626
SVM^{light}	98.71%	92.03%	0.3220
SVM^{perf}	88.13%	84.08%	0.0199
OCA	99.55%	83.58%	0.0637

Table 10. Results from the RCV1 dataset.

ALGORITHM	TRAINING	GENERALIZATION	TRAINING TIME
μABC	92.78%	91.03%	0.0256
ABC	92.75%	93.10%	0.0488
DE	92.72%	93.00%	**0.0154**
PSO	**93.42%**	**94.61%**	0.0402
KA	92.96%	91.28%	12.5998
SVM^{light}	99.01%	94.85%	0.2830
SVM^{perf}	96.51%	94.03%	0.0118
OCA	100.00%	88.15%	0.0704

Table 11. Results from the Real-Sim dataset.

ALGORITHM	TRAINING	GENERALIZATION	TRAINING TIME
μABC	97.86%	96.28%	0.0336
ABC	**98.25%**	**96.88%**	0.0612
DE	98.21%	96.51%	**0.0147**
PSO	98.30%	96.46%	0.0311
KA	97.99%	96.20%	12.6350
SVM^{light}	99.67%	97.63%	0.1510
SVM^{perf}	98.81%	97.28%	0.0107
OCA	99.73%	92.65%	0.0378

Table 12. Results from the Worm dataset.

ALGORITHM	TRAINING	GENERALIZATION	TRAINING TIME
μABC	81.60%	80.30%	0.0268
ABC	79.10%	77.77%	**0.0178**
DE	**82.81%**	**81.70%**	0.0201
PSO	81.01%	80.41%	0.0275
KA	80.86%	79.43%	12.6125
SVM^{light}	97.79%	95.35%	0.3150
SVM^{perf}	99.86%	93.80%	0.0200
OCA	100%	89.00%	0.0840

Table 13. Training accuracy results for PSP uniform frequency subsets.

ALGORITHM	$\Omega = 7$	$\Omega = 8$	$\Omega = 9$
μABC	74.23%	74.88%	73.83%
ABC	74.24%	75.36%	74.40%
DE	74.77%	**75.76%**	74.29%
PSO	**75.23%**	75.35%	**74.94%**
SVM^{light}	86.98%	87.95%	88.40%
OCA	100.00%	100.00%	100.00%
SVM^{perf}	100.00%	100.00%	100.00%
KA	75.16%	76.08%	75.18%

Table 14. Training time results for PSP uniform frequency subsets.

ALGORITHM	$\Omega = 7$	$\Omega = 8$	$\Omega = 9$
μABC	0.0297s	0.0304s	0.0224s
ABC	0.0084s	0.0060s	0.0055s
DE	**0.0054s**	**0.0041s**	**0.0026s**
PSO	0.0065s	0.0056s	0.0059s
SVM^{light}	0.0210s	0.0220s	0.0200s
OCA	0.1978s	0.2039s	0.3558s
SVM^{perf}	0.2910s	0.1820s	0.2900s
KA	1.2851s	1.2811s	1.2811s

Table 15. Cross-validation accuracy results for PSP uniform frequency subsets.

ALGORITHM	$\Omega = 7$	$\Omega = 8$	$\Omega = 9$
μABC	73.55%	73.08%	72.40%
ABC	73.75%	**74.80%**	72.93%
DE	**74.43%**	74.63%	73.08%
PSO	**74.43%**	74.05%	**73.58%**
SVM^{light}	73.33%	70.87%	70.68%
OCA	66.82%	65.28%	64.12%
SVM^{perf}	66.90%	65.37%	64.22%
KA	74.20%	75.15%	73.80%

Table 16. Training accuracy results for PSP uniform length subsets.

ALGORITHM	$\Omega = 7$	$\Omega = 8$	$\Omega = 9$
μABC	**74.86%**	**73.33%**	74.60%
ABC	73.70%	70.95%	74.07%
DE	74.72%	72.23%	75.32%
PSO	74.31%	71.19%	**75.99%**
SVM^{light}	88.05%	91.45%	88.80%
OCA	100.00%	100.00%	100.00%
SVM^{perf}	99.95%	100.00%	100.00%
KA	74.53%	71.04%	74.43%

Table 17. Training time results for PSP uniform length subsets.

ALGORITHM	$\Omega = 7$	$\Omega = 8$	$\Omega = 9$
μAB	0.0338s	0.0203s	0.0101s
ABC	0.0064s	0.0058s	**0.0024s**
DE	**0.0059s**	**0.0040s**	**0.0024s**
PSO	0.0075s	0.0066s	0.0050s
SVM^{light}	0.0200s	0.0180s	0.0250s
OCA	0.1830s	0.1600s	0.1931s
SVM^{perf}	0.1950s	0.1080s	0.1730s
KA	1.2963s	1.2827s	1.2690s

Table 18. Cross-validation accuracy results for PSP uniform length subsets.

ALGORITHM	$\Omega = 7$	$\Omega = 8$	$\Omega = 9$
μABC	**74.03%**	**71.53%**	73.55%
ABC	72.35%	69.85%	72.58%
DE	72.78%	70.48%	74.03%
PSO	73.25%	69.55%	**75.05%**
SVM^{light}	72.00%	72.34%	72.60%
OCA	64.28%	68.34%	67.05%
SVM^{perf}	64.38%	68.41%	67.13%
KA	73.15%	69.40%	72.88%

Table 19. Roc curve areas obtained from large-scale datasets.

DATASET	AREA
Astro-Ph	0.9762
Aut-AVN	0.9803
C11	0.9160
CCAT	0.9292
RCV1	0.9590
Real-Sim	0.9865
Worm	0.9891

Table 20. Roc curve significance level obtained from large-scale datasets.

	AUT-AVN	C11	CCAT	RCV1	REAL-SIM	WORM
Astro-Ph	0.9992	0.918	0.9275	0.9483	0.9817	0.9721
Aut-AVN		0.9206	0.9303	0.9537	0.9851	0.9775
C11			0.9867	0.9471	0.9108	0.906
CCAT				0.9595	0.9194	0.914
RCV1					0.9351	0.9246
Real-Sim						0.9925

phase is not the strongest point of the algorithm. Notably, in terms of training and generalization, our approach shows similar or better results to the ones obtained by the original KA algorithm, but in a fraction of the time. The best global results shown on Tables 7 to 18 are underlined, and the best results obtained by our approach are written in bold letters.

As can be seen from the ROC curves in Figures 6A to 6G and in Table 19, the generalization performances of the classifiers shown in this paper are very similar (the curves overlap each other) with excellent values for area under the curve (AUC), ranging from 0.9160 to 0.9891. The ROC curves for the PSP dataset (Table 21 and Figs. 7A to 7F) gave good values for AUC, ranging from 0.8087 to 0.8345. Even though the generalization tests performed on the Worm dataset are not as good as the rest of the generalization tests, it gave the best AUC result compared to the other ROC curve results.

To detect diffierences between solvers across multiple test attempts, Matlab's™ implemention of the Friedman test was applied to the results of the four solvers that gave the best generalization results (DE, PSO, SVM^{light} and SVM^{perf}).[78] For the test an $\alpha = 0.05$ was used with 3 degrees of freedom, using as null hypothesis H_0 the statement that there is no difference between the classifiers, and as alternative hypothesis H_1 the statement that there is a difference. According to the $\mathcal{X}^2$ table, if our $\mathcal{X}^2$ value is greater than 7.815, the null hypothesis will be rejected. The results obtained from the tests were:

- Friedman test applied to the datasets shown in Table 1: $\mathcal{X}^2 = 5.9$, a value smaller than 7.815, with P-value = 0.1161, which is greater than 0.05. From the results shown in Table 4A, we can state that hypothesis H_0 is supported. The Tukey test was used to test which classifiers are statistically significant to one another.[79] From the test we obtained an honest significant difference of 2.95; when this value is compared to the results presented in Table 5A, it is easy to see that there is no statistically significant difference between the solvers, since the difference between each pair of means is less than this value.
- Friedman test applied to the PSP dataset: $\mathcal{X}^2 = 13.4$, a value greater than 7.815, with P-value = 0.0037, which is smaller than 0.05. From the results shown in Table 4B, we can state that hypothesis H_0 is rejected.

From the test we obtained an honest significant difference of 1.17; when this value is compared to the results presented in Table 5B it is apparent that there is a statistically significant difference between SVM^{perf} and the rest of the solvers. This is easily noticed since SVM^{perf} gave the worst results in the cross-validation tests for the PSP dataset.

Every possible pair of ROC curves obtained from the datasets shown in Table 11 was compared using MedCalc© to obtain their significance level. From the results shown in Table 20 it can be stated that hypothesis H_0 is accepted in all the cases. The same procedure was applied to the ICOS PSP dataset. The results presented in Table 22 also support the H_0 hypothesis.

Conclusions

We developed a simple-to-implement method for classifying sparse, largescale datasets using parallelism with four EA.

Table 21. Roc curve areas obtained from ICOS PSP dataset.

UNIFORM:	Ω	AREA
Length	7	0.8134
	8	0.8345
	9	0.8242
Frequency	7	0.8248
	8	0.8229
	9	0.8087

Table 22. Roc curve significance level obtained from ICOS PSP dataset (where UF is uniform frequency and UL is uniform length).

	UF8	UF9	UL7	UL8	UL9
UF7	0.9988	0.99	0.9934	0.9945	0.9996
UF8		0.9922	0.9951	0.9941	0.9993
UF9			0.9975	0.9866	1.0099
UL7				0.9896	0.9943
UF8					0.9947

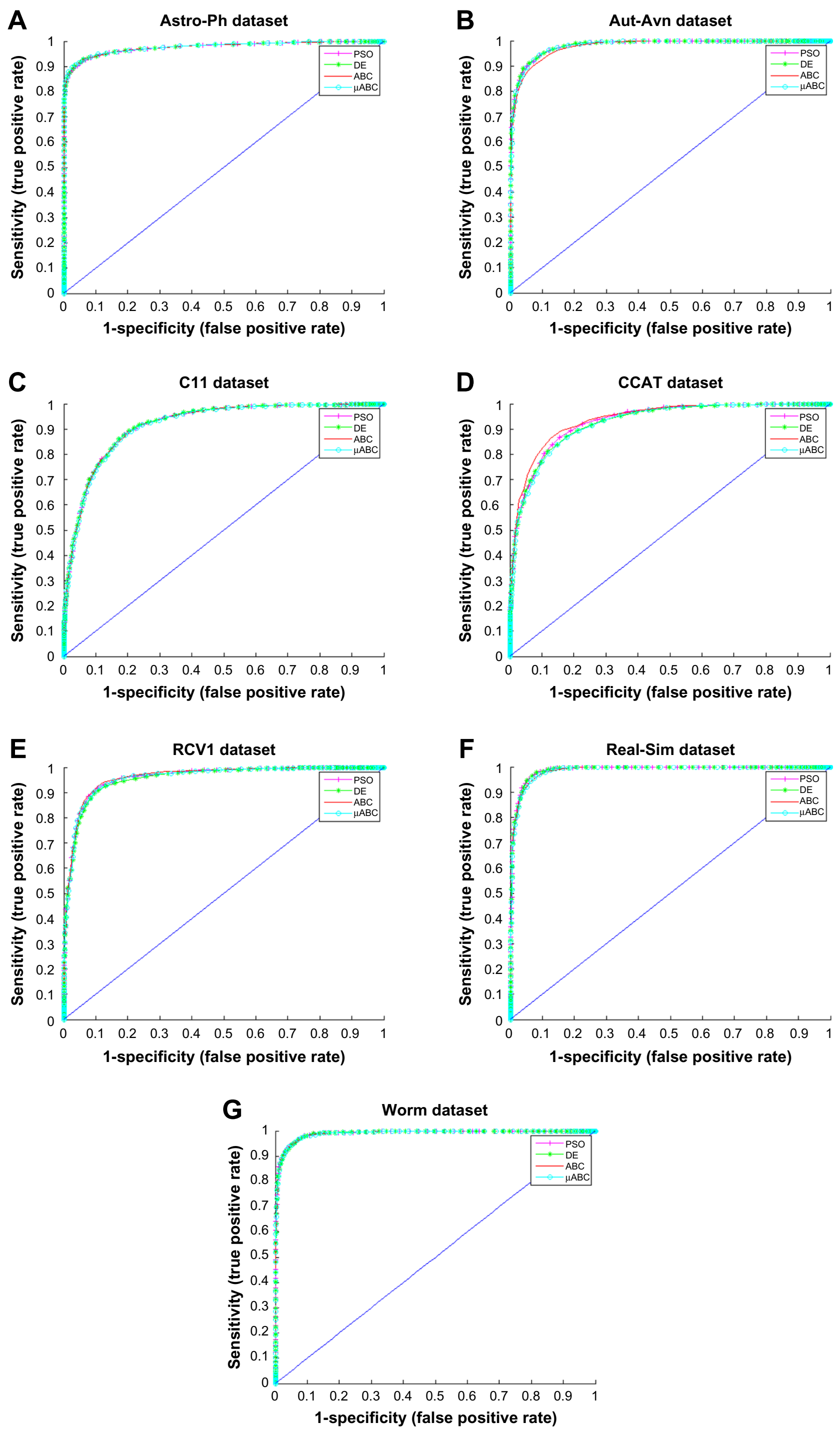

Figure 6. ROC curves obtained from large-scale datasets.

As can be seen in the results, the approach also works for classifying not-so-sparse data in very short amounts of time without increasing the complexity of the algorithm. Even though the approach did not give good results in the training phase, it gave good generalization results in competitive or smaller amounts of time compared with those obtained by algorithms such as KA, OCA, SVM^{light} and SVM^{perf} for classifying several datasets from different areas and PS data. The simplicity of the EA and training function makes it easier to implement and parallelize the approach. From the Friedman test it can be concluded that there is no difference in terms of generalization between the approaches that use PSO and DE, compared to SVM^{light}. The Tukey test confirms that there is no statistically significant difference between the three algorithms, from which it can be concluded that they have the same generalization capabilities. The ROC curve comparisons also show that the algorithms' ranges from good to excellent, since the area under the curve is greater than 0.8. These results combined with the simplicity and lineal complexity of the algorithms is what makes this

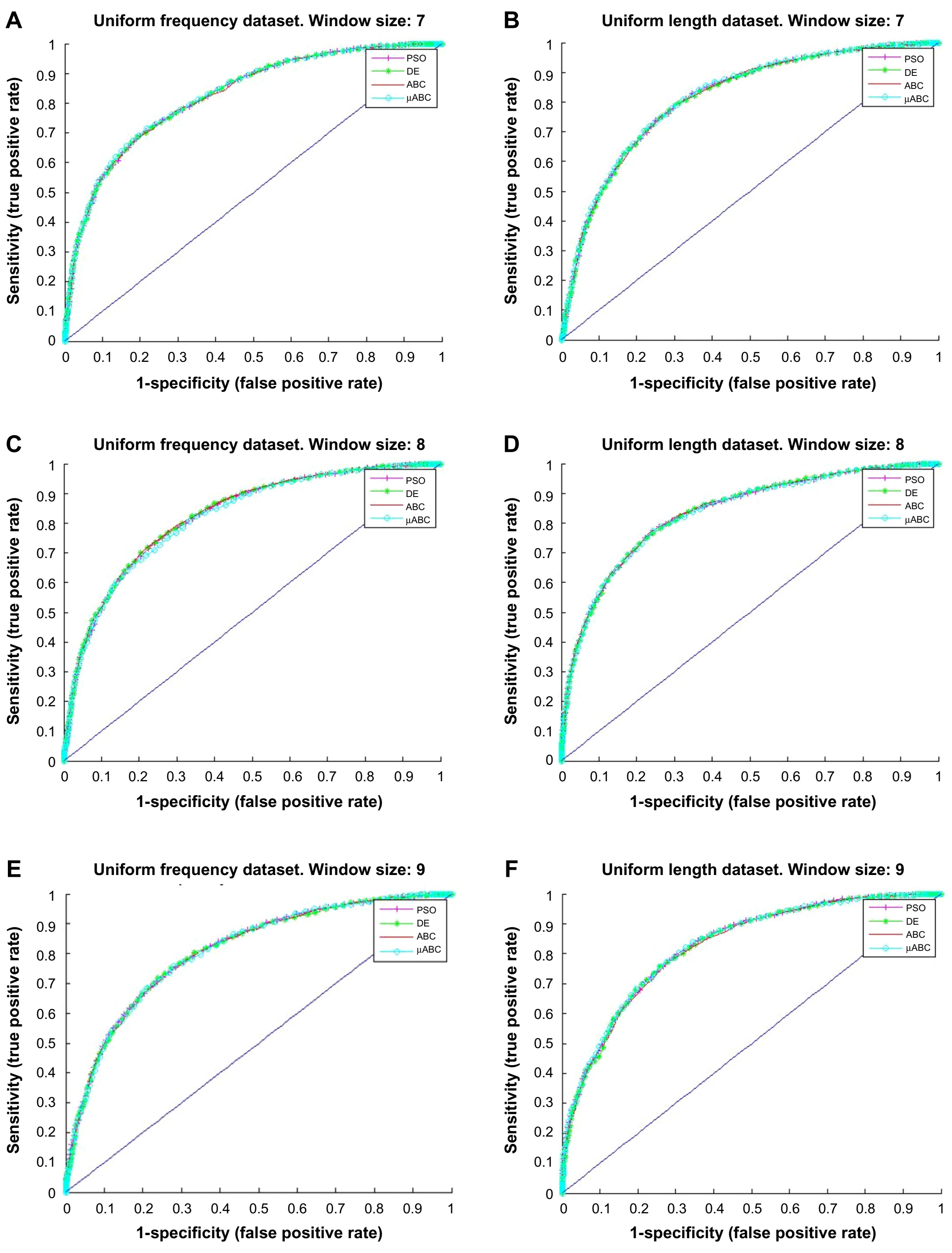

Figure 7. ROC curves obtained from the ICOS PSP dataset.

approach an appealing algorithm to be used on large-scale classification problems.

Comparing the four EAs using variants proposed, it is easy to notice that the *DE* version is the fastest and also has a good generalization capability; future improvements of the method will focus on the DE approach.

Future work includes a multiclass version of this approach, an implementation of the algorithm that can run in computer clusters, and improvements to the accuracy of the training capability of the algorithms.

Acknowledgment

We are very thankful to the projects SEP CONACYT CB-258068 and CB-256769 for supporting this work.

Author Contributions

Conceived and proposed the initial idea of the work: NAD. Conceived and designed the experiments: NAD, AAG, JM, ALF, CLF. Analyzed the data: NAD, AAG, JM, ALF, AYA. Wrote the first draft of the manuscript: NAD, AAG, AYA, CLF. Contributed to the writing of the manuscript:

NAD, AAG, AYA, CLF. Agree with manuscript results and conclusions: NAD, AAG, CLF, AYA, JM, ALF. Jointly developed the structure and arguments for the paper: NAD, AAG, CLF, AYA, JM, ALF. Made critical revisions and approved final version: NAD, AAG, CLF, AYA, JM, ALF. All authors reviewed and approved of the final manuscript.

REFERENCE

1. Kao J, Chuang J, Wang T. Chromosome classification based on the band profile similarity along approximate medial axis. *Pattern Recognition*. 2008;41:77–89.
2. Fawcett T. “In vivo” spam filtering: A challenge problem for KDD. *SIGKDD Explorations Newsletter*. 2003;5:140–8.
3. Bottou L, Chapelle O, DeCoste D, Weston J. *Large-scale kernel machines*. Neural Information Processing. MIT Press 2007.
4. Menon AK. Large-scale support vector machines: Algorithms and theory. *Research Exam*, *University of California*, *San Diego*. 2009: 1–17.
5. Frieß TT, Cristianini N, Campbell C. The Kernel-Adatron algorithm: A fast and simple learning procedure for support vector machines. In *Proceedings of the Fifteenth International Conference on Machine Learning* Morgan Kaufmann 1998.
6. Cortes C, Vapnik V. Support vector networks. *Machine Learning*. 1995;20:273–97.
7. Osowski S, Siwek K, Markiewicz T. MLP and SVM networks: A comparative study. In *Proceedings of the 6th Nordic Signal Processing Symposium*. 2004:9–11.
8. Lee Y, Kim MW, Song HK, Park SC. Analysis of classification performance for Hopfield network with predefined correlated exemplar patterns. In *International Joint Conference on Neural Networks*. 1990;1:803–8.
9. Lippmann RP. Pattern classification using neural networks. *Communications Magazine*. 1989;27:47–50.
10. Kivinen J, Smola AJ, Williamson RC. Online learning with kernels. *Transactions on Signal Processing*, *IEEE*. 2004;52:2165–76.
11. Shalev-Shwartz S, Nathan S. SVM optimization: Inverse dependence on training set size. In *Proceedings of the 25th International Conference on Machine Learning*: 2008:928–35.
12. Teo CH, Smola A, Vishwanathan SVN, Le QV. A scalable modular convex solver for regularized risk minimization. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2007: 727–36.
13. Franc V, Sonnenburg S. Optimized cutting plane algorithm for large-scale risk minimization. *Journal of Machine Learning Research*. 2009;10:2157–92.
14. Haykin S. *Neural networks*: *A comprehensive foundation* Prentice-Hall 3ed. 2007.
15. Müller KR, Mika S, Rätsch G, Tsuda K, Schölkopf B. An introduction to kernel-based learning algorithms. *IEEE Transactions on Neural Networks*. 2001;12:181–201.
16. Shalev-Shwartz S, Singer Y, Nathan S. Pegasos: Primal estimated subgradient solver for SVM. In *Proceedings of the 24th International Conference on Machine Learning*. 2007:807–14.
17. Bordes A, Bottou L, Gallinari P. SGD-QN: Careful quasi-Newton stochastic gradient descent. *Journal of Machine Learning Research*. 2009;10:1737–54.
18. Vavasis SA. Complexity theory: Quadratic programming. In *Encyclopedia of Optimization*. Springer, US; 2009:451–4.
19. Garcia E, Rangel P, Lozano F. Adaptive support vector machines for time series prediction. In *Proceedings of XI Simposio de Tratamiento de Señales*, *Imagenes y Vision Artificial*. 2006.
20. Floudas CA, Visweswaran V. *Quadratic optimization*; Springer, US;2:1995.
21. Frasch JV, Sager S, Diehl M. A parallel quadratic programming method for dynamic optimization problems. *Mathematical Programming Computation*. 2013:1–41.
22. Mierswa I. Evolutionary learning with kernels: A generic solution for large margin problems. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation (GECCO '06)*: ACM Press, New York, NY, USA; 2006: 1553–60.
23. Lessmann S, Stahlbock R, Crone SF. Genetic algorithms for support vector machine model selection. In *In International Joint Conference on Neural Networks*. 2006:3063–9.
24. Adankon MM, Cheriet M. Genetic algorithm-based training for semi-supervised SVM. *Neural Computing and Applications*. 2010;19:1197–206.
25. Huaitie X, Guoyu F, Zhiyong S, Jianjun C. Hybrid optimization method for parameter selection of support vector machine. In *IEEE International Conference on Intelligent Computing and Intelligent Systems (ICIS)*. 2010;1:613–6.
26. Sudholt D. Parallel evolutionary algorithms. In *Handbook of Computational Intelligence*. Springer;2015:929–59.
27. Karaboga D, Gorkemli B, Ozturk C, Karaboga N. A comprehensive survey: Artificial bee colony (ABC) algorithm and applications. *Artificial Intelligence Review*. 2014;42: 21–57.
28. Das S, Suganthan PN. Differential evolution: A survey of the state-of-the-art. *IEEE Transactions Evolutionary Computation*. 2011;15:4–31.
29. Rajasekhar A, Das S, Das S. μABC: A micro artificial bee colony algorithm for large-scale global optimization. In *GECCO'12 Companion*: ACM; 2012:1399–400.
30. Arana-Daniel N, Gallegos A, Lopez-Franco C, Alanis AY. Smooth global and local path planning for mobile robot using particle swarm optimization, radial basis functions, splines and Bézier curves. In 2014 *IEEE Congress on Evolutionary Computation (CEC)*: 175–82 IEEE 2014.
31. Yoshida H, Kawata K, Fukuyama Y, Takayama S, Nakanishi Y. A particle swarm optimization for reactive power and voltage control considering voltage security assessment. *Power Systems*, *IEEE Transactions on*. 2000;15:1232–9.
32. Noman N, Iba H. Differential evolution for economic load dispatch problems. *Electric Power Systems Research*. 2008;78:1322–31.
33. Karaboga D, Akay B, Ozturk C. Artificial bee colony (ABC) optimization algorithm for training feed-forward neural networks. In *Modeling Decisions for Artificial Intelligence*: Springer; 2007:318–29.
34. Indiveri G. *Handbook of computational intelligence* Berlin, Heidelberg: Springer-Verlag;2015.
35. Branden CI. *Introduction to protein structure* Garland Science 1999.
36. Westhead DR, Thornton JM. Protein structure prediction. *Current opinion in biotechnology*. 1998;9:383–9.
37. Skolnick J, Zhang Y. Protein structure prediction. *Systems Biology*. 2007;1: 187–218.
38. Marks DS, Hopf TA, Sander C. Protein structure prediction from sequence variation. *Nature Biotechnology*. 2012;30:1072–80.
39. Hegyi H, Gerstein M. The relationship between protein structure and function: A comprehensive survey with application to the yeast genome. *Journal of molecular biology*. 1999;288:147–64.
40. Rangwala H, Karypis G. *Introduction to protein structure prediction* John Wiley & Sons, Inc. New York 2010.
41. Sta¸por K. Using machine learning approach for protein fold recognition. *Studia Informatica*. 2011;32.
42. Hill JR, Kelm S, Shi J, Deane CM. Environment specific substitution tables improve membrane protein alignment. *Bioinformatics [ISMB/ECCB]*. 2011;27:15–23.
43. Shamim MTA, Anwaruddin M, Nagarajaram HA. Support vector machine-based classification of protein folds using the structural properties of amino-acid residues and amino-acid residue pairs. *Bioinformatics*. 2007;23:3320–7.
44. Kelley LA, MacCallum RM, Sternberg MJE. Enhanced genome annotation using structural profiles in the program 3D-PSSM. *Journal of Molecular Biology*. 2000;299:501–22.
45. Shi J, Blundell TL, Mizuguchi K. FUGUE: Sequence-structure homology recognition using environment-specific substitution tables and structure-dependent gap penalties. *Journal of Molecular Biology*. 2001;310:243–57.
46. Jones DT, Taylort WR, Thornton JM. A new approach to protein fold recognition. *Nature*. 1992;358:86–9.
47. Ho HK, Zhang L, Ramamohanarao K, Martin S. *Protein supersecondary structures*: *A survey of machine learning methods for secondary and supersecondary protein structure prediction*.: Totowa, NJ: Humana Press; 2013:87–106.
48. Mandle AK, Jain P, Shrivastava SK. Protein structure prediction using support vector machine. *International Journal on Soft Computing*. 2012;3:67.
49. Cai YD, Liu XJ, Biao Xu X, Zhou GP. Support vector machines for predicting protein structural class. *BMC Bioinformatics*. 2001;2:3.
50. Wang S, Peng J, Ma J, Xu J. Protein secondary structure prediction using deep convolutional neural fields. *Scientific reports*. 2016;6.
51. Hue M, Riffle M, Vert JP, Noble WS. Large-scale prediction of protein-protein interactions from structures. *BMC bioinformatics*. 2010;11:1.
52. Cai YD, Liu XJ, Xu X, Chou KC. Prediction of protein structural classes by support vector machines. *Computers & Chemistry*. 2002;26:293–6.
53. Chen C, Zhou X, Tian Y, Zou X, Cai P. Predicting protein structural class with pseudo-amino acid composition and support vector machine fusion network. *Analytical biochemistry*. 2006;357:116–21.
54. Guo J, Chen H, Sun Z, Lin Y. A novel method for protein secondary structure prediction using dual-layer SVM and profiles. *Proteins*: *Structure*, *Function*, *and Bioinformatics*. 2004;54:738–43.
55. Li ZC, Zhou XB, Lin YR, Zou XY. Prediction of protein structure class by coupling improved genetic algorithm and support vector machine. *Amino Acids*. 2008;35:581–90.
56. Anlauf JK, Biehl M. The Adatron: An adaptive perceptron algorithm. *Europhysics Letters*. 1989;10:687.
57. Opper M. Learning times of neural networks: exact solution for a perceptron algorithm. *Physical Review A*. 1988;38:3824.
58. Dianati M, Song I, Treiber M. An Introduction to Genetic Algorithms and Evolution Strategies. tech. rep. University of Waterloo, Ontario, N2 L 3G1, Canada; 2002.
59. Simon D. *Evolutionary optimization algorithms* Wiley 2013.
60. Shi Y, Eberhart RC. Empirical study of particle swarm optimization. In *Proceedings of the* 1999 *Congress on Evolutionary Computation*. 3:1950–99.

61. Benala TR, Jampala SD, Villa H, Konathala B. A novel approach to image edge enhancement using artificial bee colony optimization algorithm for hybridized smoothening filters. In *Nature & Biologically Inspired Computing*: IEEE;2009: 1071–6.
62. Fraga LGL. Self-calibration from planes using differential evolution. In *Progress in Pattern Recognition*, *Image Analysis*, *Computer Vision*, *and Applications*: 724–31 Springer 2009.
63. AlRashidi MR, El-Hawary ME. A survey of particle swarm optimization applications in electric power systems. *IEEE Transactions on Evolutionary Computation*. 2009;13:913–8.
64. Karaboga D. An idea based on honey bee swarm for numerical optimization. tech. rep. Engineering Faculty, Computer Engineering Department 2005.
65. Storn R, Price K. *Differential evolution*: *A simple and efficient adaptive scheme for global optimization over continuous spaces*; ICSI Berkeley; 3;1995.
66. Kennedy J, Eberhart RC. Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks*; IEEE 1995;4:1942–8.
67. Parsopoulos KE, Vrahatis MN. *Particle swarm optimization and intelligence*: *Advances and applications*. IGI Global; 2010.
68. Frieß TT, Harrison R. The Kernel-Adatron with bias unit: Analysis of the algorithm. 1998.
69. Bacardit J, Krasnogor N. The ICOS PSP benchmarks repository. 2008. (http://icos.cs.nott.ac.uk/datasets/psp benchmark.html).
70. Stout M, Bacardit J, Hirst JD, Krasnogor N. Prediction of recursive convex hull class assignments for protein residues. *Bioinformatics*. 2008;24:916–23.
71. Ramanna S, Jain LC, Howlett RJ. *Emerging paradigms in machine learning*. Smart Innovation, Systems and Technologies Springer Berlin Heidelberg 2012.
72. Liu H, Hussain F, Tan C L, Dash M. Discretization: An enabling technique. *Data mining and knowledge discovery*. 2002;6:393–423.
73. Kinjo AR, Horimoto K, Nishikawa K. Predicting absolute contact numbers of native protein structure from amino acid sequence. *Proteins*: *Structure*, *Function*, *and Bioinformatics*. 2005;58:158–65.
74. Joachims T. Training linear SVMs in linear time. In *Proceedings of the Conference on Knowledge Discovery and Data Mining* 2006.
75. Sanderson C. Armadillo: An open source C++ linear algebra library for fast prototyping and computationally intensive experiments. tech. rep. 2010.
76. Joachims T. Making large scale SVM learning practical. In *Advances in Kernel Methods – Support Vector Learning* MIT Press 1999.
77. Platt J. Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines. 1998.
78. Friedman M. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the american statistical association*. 1937;32:675–701.
79. Tukey JW. Comparing individual means in the analysis of variance. *Biometrics*. 1949:99–114.