CHAPTER 5

CLASSIFIER OPTIMISATION

The parameters of classification algorithms have a profound influence on their performance in terms of generalization ability and robustness to noise. For an SVM-based classifier, the kernel function plays a vital role in dealing with non-linear and arbitrarily structured data, whose parameters also have an impact on the classification performance.

Any attempt to improve the performance of an SVM based classifier must invariably include parameter optimisation. Also, underwater target classification with dynamically changing constraints, necessitates dynamic selection of the optimal algorithmic parameters, kernel function and kernel parameters, which can be achieved through various optimisation techniques. The idea is to find out the parameters that maximize the performance of the proposed classifier. This chapter throws light upon the different optimisation strategies that can be adopted, associated concepts, and a brief about their merits and demerits. The chapter mainly focuses on meta-heuristic optimization algorithms, and also the different meta-heuristic optimisation algorithms adopted in this work.

5.1 Background - The Optimisation problem

Solutions to problems may not always have a binary nature, but are often rated in terms of quality concerning a performance metric. Learning algorithms, usually depend on parameters which control the size of the search
Chapter 5 Classifier Optimisation

space or the way the search is conducted in the search space. Optimisation is performed to select the one which is of the best quality from a set of many candidate solutions that differ in quality, under the given circumstances. Optimisation reflects from the analytical quality of all intelligent beings to pursue the best. The mathematical formulation of the concept of optimisation can be as follows. Consider a set of candidate solutions, $X$, subjected to the optimisation problem. Typically, $X$ is $n$-dimensional over certain domain, referred to as the search space. The optimisation problem is defined by an objective function (also referred to as cost function or fitness function), to estimate the performance of the candidate in $X$ on the given problem.

$$f : X \rightarrow \mathbb{R}$$  \hspace{1cm} 5.1

The objective function, typically expressed as a function of design variable, defines a criteria to compare possible solutions. The optimisation problem can be formulated with an objective to find the best candidate solution that minimises or maximizes the fitness function $f$, as follows

$$\text{find } p \in X, \text{ so that } \forall o \in X : f(p) \leq f(o)$$  \hspace{1cm} 5.2

In practical applications, solutions to this problem may be a subset of all possible combinations or permutations of the elements of the vector $X$. Such problems are characterised by a finite set of solutions and are referred to as combinatorial optimisation problems.

5.2 Classification of Optimisation Problems

Optimisation problem can be classified in several ways as follows

- Constrained and Unconstrained: Constrained optimisation problems are subjected to one or more constraints and hence maintain the
search effort within a feasible region, whereas unconstrained optimisation problems have no constraints at all.

- Single objective and multi-objective: Optimisation problems may be based on a single objective function or multiple objective functions which involves decision making with multiple criteria.

- Nature of objective function and constraints: Optimisation problems can be classified as linear, quadratic, polynomial and non-linear depending upon the nature of the objective functions and the constraints. Determination of the type of solution often depends on the nature of involved functions.

- Deterministic and Stochastic: In deterministic optimisation problem, all the design variables are deterministic. In stochastic optimisation problem some or all the parameters are stochastic (non deterministic or probabilistic).

### 5.3 Curse of Dimensionality

The only way to ensure global optimality in solutions of an optimization problem is to evaluate all the candidate solutions, which is often computationally intractable. The adversity exponentially grows worse with increasing dimensionality, which is termed as the curse of dimensionality. Hence, it is essential to adopt optimisation methods that give acceptable performance in fewer dimensions and avoid exponentially increasing number of fitness evaluations with increasing dimensions. The optimisation methods should preferably have a linear relationship between the dimensionality of the problem and the number of candidate solutions i.e. it should have linear time complexity $O(n)$ in the dimensionality $n$ of the problem to be optimised.
Chapter 5 Classifier Optimisation

5.4 **NP Theory**

NP theory encompasses concepts that explain the solvability of an algorithm. An optimisation algorithm can be considered as efficient if it has polynomial run time [162]. The problems solved in polynomial time, by a deterministic machine is referred to as P problem while by non-deterministic machines is referred as NP (non-deterministic polynomial). The solutions to these problems are also verifiable in polynomial time.

NP-hardness (non-deterministic polynomial-time hardness), in computational complexity theory, is the defining property of a class of problems that are, informally, "at least as hard as the hardest problems in NP". More precisely, a problem H is NP-hard when every problem L in NP can be reduced in polynomial time to H; that is, assuming a solution for H takes 1 unit time, we can use H's solution to solve L in polynomial time. As a consequence, finding a polynomial algorithm to solve any NP-hard problem would give polynomial algorithms for all the problems in NP, which is unlikely as many of them are considered hard.

5.5 **No Free Lunch Theorem**

The No Free Lunch (NFL) theorem proposed by Wolpert and Macready [163] states that ‘if an algorithm performs well in a certain class of problems, then it necessarily pays for that with degraded performance on the set of all remaining problems’. Consider \( f: S \rightarrow Y \) which is the pair of all possible mappings within the range of values \( Y_s \) in the solution space \( S \).

Then \( T_m \) is defined as

\[
T_m = \{(s_1, y_1), \ldots, (s_m, y_m)\} \quad \text{where} \quad y_j = f(s_j) \quad 5.3
\]

NFL theorem states that for any pair of algorithms \( a \) and \( b \),
where $\mathcal{P}(X^O, \hat{O}, \hat{\mathcal{P}})$ is the probability of obtaining a certain sequence of values $y_1, y_2, y_3, \ldots, y_m$ given that the search is for the function $f$ with algorithm $a$.

The NFL theorem suggests that any two algorithms are equivalent when their performance is averaged across all problems, and hence, any one optimisation method will be as likely as any other to find a satisfactory solution. Thus for any optimisation algorithm, high performance over one class of problems is offset by performance over another class and a universally best method does not exist as illustrated in Figure 5.1. There may exist an optimisation method specialised for a particular problem. There are also algorithms which deliver good results for many different problem classes, but may be outperformed by a highly specialised method in each of them.

Another manifestation of NFL theorem is related to the objective of an optimisation problem. The objective function of an optimisation problem is typically formulated based on the expected physical outcome as well as economic considerations. Following the NFL theorem, an algorithm which
performs well for a particular objective, say speed, may perform poorly for another metric say, accuracy. Thus, there always exists a fundamental trade-off between performance vs cost or performance vs reliability aspects of an optimisation algorithm and no choice is universally better than any other [115].

5.6 Optimisation Search Strategies

A large number of optimisation algorithms have been developed based on different search strategies. The naïve strategy towards finding the optimal parameters is to conduct an exhaustive search in the parameter space, which guarantees to find an optimal solution at a problem-specific computational cost. Since many practical optimisation problems are NP-hard, no polynomial-time algorithm is known, and the computational cost for exhaustive search is prohibitively high since it would correspond to running the algorithm for every possible value of parameters and hence the ideal linear time complexity $O(n)$ described in section 5.4 cannot be attained.

The traditional way of parameter optimisation is to conduct a grid search, a brute force method which exhaustively searches through the subset of parameters of the learning algorithm, guided by some performance metric. Grid search involves dividing the parameter space into grids of uniform size. The model is trained for the parameter values at each grid locations and the response is evaluated through a suitable performance metric, to obtain the optimal parameter set.

Another method of parameter optimisation is to employ greedy search techniques, which makes a greedy choice at each step for a solution that yields maximum performance, with the hope of finding a global optimum, without assessing its consequences. These techniques have high chances of
getting trapped in local optima. Greedy search techniques work well for monotonous objective function with smooth solution spaces [164]. However, for multimodal objectives, it is likely to get stuck in local optima. This approach is also called hill-climbing approach, referring to a mountaineer who chooses every next step in a manner that yields maximal improvement. Gradient search algorithms are examples of greedy search techniques.

Stochastic, heuristic and meta-heuristic optimisation techniques are resorted to, in applications where exhaustive search techniques seem to be impractical. Stochastic optimisation techniques are applied to solve highly non-linear, high-dimensional data difficult to be solved by classical deterministic methods. Stochastic optimisation techniques generate and use random variables for the formulation of the optimisation problem which involve random objective functions or random constraints. These techniques are faster than exhaustive search techniques but cannot guarantee best solutions or global solutions. Heuristic optimisation techniques are experience based techniques and are employed to solve complex logistics problems of higher dimension [165]. Heuristic derives from the Greek verb *heuriskein* means ‘to find’ by trial and error, and heuristic algorithms start searching the solution space with an initial guess and try to improve the quality of solutions over the course of iterations. Heuristic algorithms are capable of finding satisfactory solutions at a faster rate, but do not always guarantee to find optimal solutions.

Meta-heuristic algorithms are strategies that guide the search process. ‘Meta’ means ‘beyond’ or ‘higher level’, and the meta-heuristic algorithm guides a subordinate heuristic search through a combination of intelligent randomisation and local search, or in other words, exploration and exploitation. “A meta-heuristic is formally defined as an iterative generation process which guides a subordinate heuristic by combining intelligently
different concepts for exploring and exploiting the search space; learning strategies are used to structure information to efficiently find near-optimal solutions [166].” Meta-heuristics is an algorithmic architecture that customises the algorithmic procedures of a more problem-specific subordinate heuristics to obtain high-quality solutions. The subordinate heuristics may be a high-level or low-level procedure ranging from a simple local search to a complex search procedure.

Exploration (randomisation/diversification) and exploitation (intensification) are the two competing driving forces of a meta-heuristic algorithm, which allows it to effectively and efficiently explore the search space. Exploration and exploitation of the search space, are two competing design goals, in which the algorithm must be ‘clever’ to intensively exploit areas of the search space with high-quality solutions, as well as move to unexplored search spaces when required. Exploration increases the diversity of solutions and prevents them from being trapped at local optima, while exploitation in promising areas of the search space, based on accumulated search experience will increase the chances of the algorithm to find optimal solutions. A good combination of exploration and exploitation usually ensure global optimality [166].

Nature has evolved over millions of years and has found optimal solutions to a variety of natural phenomena. Nature-inspired meta-heuristic algorithms mimic different strategies of nature in finding optimal solutions. They can be grouped in four main categories: evolution-based, physics-based, swarm-based and human-based algorithms as depicted in Fig. 5.2 [167].

Evolution-based algorithms, inspired by the laws of natural evolution, arrive at an optimal solution by optimising a randomly generated
population of possible solutions, over the course of iterations. The most popular evolution-inspired technique is Genetic Algorithms (GA) that simulates the evolutionary laws of nature. Physics-based meta-heuristic algorithms are formulated by imitating the physical laws in the universe. The most popular algorithms in this category are Simulated Annealing (SA) and Gravitational Local Search (GLS).

Swarm-based techniques are formulated by abstracting the social behaviour of groups of animals or birds. The most popular algorithm is Particle Swarm Optimisation, which inspired by the social behaviour of bird flocking or fish schooling. Human based algorithms are formulated by modelling the human behaviour. An example of human based algorithm is the brain storm (BS) optimisation algorithm which is inspired by the human brainstorming process.

5.7 Parameter Optimisation

Machine learning algorithms have behavioural parameters which have profound effects on their performance. These parameters values are heavily problem dependent in the sense that, a set of values might work best on a certain problem, well on certain other problem instances and bad on another class of instances. Hence, setting the right algorithmic parameters is an influential design goal in machine learning.

Attempts to improve the performance of a classifier should invariably consider estimating suitable parameters for the problem at hand. However, finding the best parameters is a consequential task and it is very ambitious to understand the effect of each parameter. Some parameters may even have effects on other parameters which make the problem all the more complicated. Therefore, in order to determine parameters that are adaptable to the classification problem, parameter optimisation algorithms are resorted.
Fig. 5.2 Classification of meta-heuristic algorithms

GA – Genetic Algorithm
ES – Evolution Strategy
GP – Genetic Programming
DE – Differential Evolution
GSA – Gravitational Search Algorithm
CSS – Charged System Search
CF – Central Force Optimisation
BBBC – Big Bang-Big Crunch
ACO – Ant Colony Optimisation
PSO – Particle Swarm Optimisation
ABS – Abaffy Broyden Spedicato
CS – Cuckoo Search
TLBO – Teaching – Learning – Based Optimisation
TS – Tabu Search
HS – Hirschberg-Sinclair
BSO – Brain Storm Optimisation
The search space for most parameter optimisation tasks is large, and hence exhaustive search techniques are normally not feasible in terms of runtime. Furthermore, the interdependency between the parameters and their effect on the algorithm’s performance is largely unknown most of the times. Meta-heuristic algorithms, which strikes a balance between exhaustive search as well as heuristics are adopted in this work for optimising the parameter values of the underlying SVM based target classifier. Five nature inspired meta-heuristics algorithms namely, genetic algorithm, bat algorithm, whale optimisation algorithm, stochastic fractal search and symbiotic organisms search algorithm have been adopted for improving the classifier performance by tuning the algorithmic parameters. We have also proposed a modified symbiotic organisms search algorithm which is found to have better performance in classifying the underwater targets of interest.

5.8 Genetic Algorithm (GA)

Genetic algorithm (GA), is a meta-heuristic search and optimisation algorithm formulated by the abstraction of the ‘survival of fittest’ among individuals over consecutive generations in natural evolution [168]. GA starts with a population of candidate solutions (represented by chromosomes) for the problem at hand, characterising the natural ecosystem and makes it evolve by iteratively applying a set of stochastic operators like selection, crossover and mutation.

The chromosomes contain genes which encode a particular trait of the individual which may take different settings called alleles. An individual is also referred to as genome and the set of genes in a genome are referred to as genotype, which is the base for an organisms phenotype or the traits and characteristics it exhibits. They use metaphors which follow the theory of
natural evolution such as selection, recombination and mutation to evolve the solution to a problem.

The population consists of a number of co-existing organisms that compete for the same resources. The distinguishing traits of organisms that are most fit and capable of gathering resources will be carried over to the next generation. The entire population of the ecosystem is said to evolve over time, to accommodate organisms that are fitter than the previous generation. Thus, the characteristics that promote survival are preserved across generations.

At the beginning of the run of a genetic algorithm a population of chromosomes is created randomly, representing different solutions to the problem at hand. A fitness measure (objective function) is defined for evaluating the chromosomes and assigns a fitness score to each chromosome to assess its ability to solve the problem at hand. The algorithm then selects two members from the current population as parents to create offspring in the next generation. There are different methods for the selection process, however the chances of being selected as parents for the next generation is proportional to their fitness scores. Offspring for the next generation are created from parents by exchanging the genes in a process called cross-over. The algorithm then mutates the bits in a chromosome depending on a predefined mutation rate. After cross-over and mutation, the off-springs are evaluated just like their parents to measure their fitness of survival in the population. Fig.5.3 depicts the flowchart of GA.
5.8.1 GA operators

5.8.1.1 Selection

Selection is the process of choosing successful solutions from the current population as parents, which mate and recombine to create off-springs for the next generation. Selection is very crucial for the convergence rate of the GA, as good parents generate better and fitter solutions. Different techniques for selection in GA include,

- Tournament selection: Several tournaments are played among individuals chosen at random from the population. The winner of the tournament is selected for next generation to become a parent. With
Chapter 5 Classifier Optimisation

sufficiently large tournament size, weak individuals have a lower probability to be selected, and thus selection pressure can be adjusted by altering the tournament size.

- Roulette wheel selection: In this method of selection, segments are created for each individual according to their fitness. A random number is generated and the individual whose segment spans the random number is selected. This technique is called so as it is homologous to a roulette wheel in which all chromosomes in the population has a slice proportional to its fitness.

- Rank selection: Rank selection is mostly employed when the individuals in the population have very close fitness values and selection by roulette wheel will have very low selection pressure towards fitter individuals resulting in a bad selection of parents. In rank selection, the population is ranked according to their fitness and the parents are selected depending on their ranking, with more preference given to higher ranked individuals.

- Steady-state selection: GA in steady state means that there are no generations, and instead of replacing children of selected parents in the next generation, two best individuals out of the two parents alongwith their children are added back into the population so that the population size remains constant.

5.8.1.2 Crossover

Crossover is the process of combining parents to produce offspring and is analogous to reproduction and biological crossover. Crossover operation has the primary responsibility of converging the search algorithm to an optimal solution. There are different techniques of crossover, some of which are detailed as in the following paragraphs.
• Single point crossover: A random crossover point is selected and the bits next to the cross-sites (tails of the parents) are swapped to create new off-springs as depicted in Fig.5.4.

Fig.5.4 Depiction of single point crossover

• N - point crossover: Also called multi-point crossover, N cross over points are selected at random and the off springs are created by combining the parents at the crossover point as depicted in Fig.5.5.

Fig.5.5 Depiction of N - point crossover

• Uniform crossover: In uniform crossover a crossover mask with binary values is created, and off springs are created by copying genes from the parents according to the values in this mask. At positions where there is a ‘1’ in the mask, genes are carried from one parent, and at positions where there is a ‘0’ in the mask, genes are carried from the other parent.

5.8.1.3 Mutation

Mutation operator, analogous to the biological mutation alters the value of one or more genes, to maintain genetic diversity across generations. Mutation allows exploration of the search space and is essential to the convergence of GA. Commonly used mutation schemes are random, swap, scramble, inversion, uniform as well as gaussian mutation.
Chapter 5 Classifier Optimisation

- Random Mutation: Bits to be mutated are selected randomly and flipped.
- Swap Mutation: Values of individual genes are swapped between two positions on the chromosome.
- Scramble Mutation: Values of a selected subset of genes are randomly scrambled or shuffled.
- Inversion Mutation: Values of a selected subset of genes are inverted.
- Uniform Mutation: The chosen gene to be mutated is replaced by a random value between a user specified upper and lower bounds.
- Gaussian Mutation: A Gaussian distributed random values is added to the chosen gene, and if it falls outside user specified bounds, the gene value is clipped

The proposed SVM based underwater classifier was optimised with GA. The fitness measure employed is F-score. The results of the experiments for different population sizes are tabulated in Table 5-1. Each run of the algorithm consists of 500 iterations, except when the algorithm is terminated when the average relative change in fitness value is stalled over 25 generations.

Table 5-1 Performance results with GA based optimisation for varying population size

<table>
<thead>
<tr>
<th>Population Size</th>
<th>10</th>
<th>15</th>
<th>20</th>
<th>25</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>F-score (class 1)</strong></td>
<td>0.60</td>
<td>0.68</td>
<td><strong>0.71</strong></td>
<td>0.71</td>
</tr>
<tr>
<td><strong>F-score (class 2)</strong></td>
<td>0.74</td>
<td>0.75</td>
<td><strong>0.76</strong></td>
<td>0.75</td>
</tr>
<tr>
<td><strong>F-score (class 3)</strong></td>
<td>0.73</td>
<td>0.74</td>
<td><strong>0.75</strong></td>
<td>0.74</td>
</tr>
<tr>
<td><strong>F-score (class 4)</strong></td>
<td>0.84</td>
<td>0.85</td>
<td><strong>0.86</strong></td>
<td>0.86</td>
</tr>
</tbody>
</table>
### 5.9 BAT Algorithm

BAT algorithm was formulated by Xin-She Yang [169] based on the echolocation behaviour of bats. Bats use sonar to detect prey and avoid obstacles by emitting ultrasonic bursts and use the time delay information between the emission and detection of the echo, variation of loudness in the echo and the time difference between their ears to visualize their surroundings. Bats emit about 10 to 20 ultrasonic sound bursts per second typically of 5 to 20 ms duration. They tend to decrease loudness and increase the rate of emission to about 200 pulses per second when they approach a prey. They are able to detect the distance and orientation of the target, distinguish between different types of prey and can even estimate the moving speed of the prey such as small insects.

The characteristics of the echolocation pulse of bats vary with species and the following approximations are adopted to idealize the echo location characteristics of bats in order to formulate the BAT algorithm.
Chapter 5 Classifier Optimisation

i. All bats use echolocation and they know the difference between food/prey and background barriers in some magical way.

ii. Bats fly randomly with velocity $v_i$ at position $x_i$ with a fixed frequency $f_{\text{min}}$ in, varying wavelength $\lambda$ and loudness $A_0$ to search for the prey. They can automatically adjust the wavelength (or frequency) of their emitted pulses and adjust the rate of pulse emission $r$ in the range $[0, 1]$, depending on the proximity of their target.

iii. Although the loudness can vary in many ways, we assume that the loudness varies from a large (positive) $A_0$ to a minimum constant value $A_{\text{min}}$.

With the above approximations, the algorithm is formulated as follows. The algorithm commences by initializing the position and velocity of a population of bats. During the course of iterations, the position, velocity and frequency of the bats are updated as,

\[
x_i(t + 1) = x_i(t) + v_i(t + 1) \tag{5.5}
\]
\[
v_i(t + 1) = v_i(t) + (x_i(t) - G_{\text{best}})Q_i \tag{5.6}
\]

where $G_{\text{best}}$ is the best solution obtained so far and $Q_i$ indicates the frequency of the $i^{th}$ bat which is updated in each course of iteration as follows,

\[
Q_i = Q_{\text{min}} + (Q_{\text{max}} - Q_{\text{min}})\beta \tag{5.7}
\]

where, $\beta$ is a random number drawn from a uniform distribution in $[0,1]$. To introduce exploitation of search space into the algorithm, a local random walk is performed around randomly picked solutions and is described in equation 5.8.

\[
x_{\text{new}} = x_{\text{old}} + \epsilon A^t \tag{5.8}
\]
where $\epsilon \in [-1, 1]$ is a random number, and $A_t = \langle A_t^i \rangle$ is the average loudness of all the bats at the time step $t$. The loudness of the bats decreases and pulse emission rate increases when the bats approach the prey which is simulated in the algorithm to ensure convergence. During the course of iterations, when the solutions are improved, the loudness and emission rates are updated by the following equations to converge to the optimal solution.

$$A_{t+1}^i = \alpha A_t^i$$
$$r_{t+1}^i = r_0^i [1 - \exp(-\gamma t)]$$

where $0 < \alpha < 1, \gamma > 0$ and as simulations proceeds and the algorithm converges, $A_t^i \to 0, r_t^i = r_0^i$ as $t \to \infty$. The pseudo-code of BAT algorithm is shown in Fig. 5.6.

BAT algorithm is in a way improvement over the particle swarm optimisation algorithm. The update of velocities and positions of the bats are similar to updating the pace and range of the swarming particle in PSO. However, an intense local search controlled by loudness and pulse rate makes BAT algorithm superior to PSO due to its effectiveness in balancing exploration and exploitation.

The proposed SVM based underwater classifier was optimised with BAT algorithm. The results of the experiments for different population sizes are tabulated in Table 5-2. Each run of the algorithm consists of 500 iterations, except when the algorithm is terminated when the average relative change in fitness value is stalled over 25 generations.
Chapter 5 Classifier Optimisation

Define pulse frequency
Initialize pulse rate, loudness
Initialize microbats (position, velocity and frequency)
Calculate the fitness and find the initial best solution

while (max no. of iteration not reached)
{
Generate new solutions by eqns 5.5, 5.6&5.7
if (rand > pulse rate)
    Select a solution among the best solutions
    and generate a local solution around the best
    solution, eqn 5.8
endif
Evaluate the new solutions by their fitness values
if (fitness has improved & solution is not too
    loud)
    decrease loudness and increase pulse
    emission rate, eqns 5.9&5.10
    accept the new solutions
endif
find the current best solution
}

Fig.5.6 BAT algorithm pseudo-code

Table 5-2 Performance results with BAT algorithm based optimisation for varying population size

<table>
<thead>
<tr>
<th>Population Size</th>
<th>10</th>
<th>15</th>
<th>20</th>
<th>25</th>
</tr>
</thead>
<tbody>
<tr>
<td>F-score (class 1)</td>
<td>0.60</td>
<td>0.65</td>
<td>0.63</td>
<td>0.63</td>
</tr>
<tr>
<td>F-score (class 2)</td>
<td>0.73</td>
<td>0.75</td>
<td>0.74</td>
<td>0.73</td>
</tr>
<tr>
<td>F-score (class 3)</td>
<td>0.72</td>
<td>0.77</td>
<td>0.75</td>
<td>0.72</td>
</tr>
<tr>
<td>F-score (class 4)</td>
<td>0.86</td>
<td>0.87</td>
<td>0.86</td>
<td>0.86</td>
</tr>
<tr>
<td>F-score (class 5)</td>
<td>0.79</td>
<td>0.85</td>
<td>0.81</td>
<td>0.81</td>
</tr>
<tr>
<td>F-score (class 6)</td>
<td>0.85</td>
<td>0.90</td>
<td>0.88</td>
<td>0.87</td>
</tr>
<tr>
<td>F-score (class 7)</td>
<td>0.75</td>
<td>0.78</td>
<td>0.77</td>
<td>0.75</td>
</tr>
</tbody>
</table>
Underwater Target Classifier with Improved Success Rate using Meta-Optimal Support Vector Machine

<table>
<thead>
<tr>
<th>F-score (class 8)</th>
<th>0.82</th>
<th>0.84</th>
<th>0.83</th>
<th>0.83</th>
</tr>
</thead>
<tbody>
<tr>
<td>F-score (class 9)</td>
<td>0.79</td>
<td>0.81</td>
<td>0.80</td>
<td>0.79</td>
</tr>
<tr>
<td>F-score (class 10)</td>
<td>0.62</td>
<td>0.67</td>
<td>0.65</td>
<td>0.65</td>
</tr>
<tr>
<td>F-score (class 11)</td>
<td>0.61</td>
<td>0.66</td>
<td>0.65</td>
<td>0.65</td>
</tr>
<tr>
<td>F-score (Average)</td>
<td>0.74</td>
<td>0.78</td>
<td>0.76</td>
<td>0.75</td>
</tr>
<tr>
<td>Overall Accuracy (%)</td>
<td>70.2</td>
<td>74.5</td>
<td>72.4</td>
<td>71.6</td>
</tr>
</tbody>
</table>

5.10 Whale Optimisation Algorithm

Whale Optimisation Algorithm proposed by S. Mirjalili and A. Lewis, is a nature inspired meta-heuristic algorithm, formulated by the abstraction of hunting behavior of humpback whales [167].

Fig.5.7 Bubble-net feeding behaviour of humpback whales

Humpback whales have a unique co-operative foraging behaviour which is popularly known as bubble-net feeding method shown in Fig.5.7, in which they use co-operative strategies to disorient and corral the fish into a bubble net that they create. They prefer to hunt school of small fish such as salmon, krill or herring. The whales gather together beneath the surface and
exhales out of their blowholes, to blow bubbles along a circle or 9 shaped path while continuing to encircle their prey and corralling the fish into the bubble net. The whales then simultaneously swim upwards with mouths wide open to feed on the trapped fish [170].

The WOA algorithm is modeled in three phases by simulating *i.e.* encircling the prey, the bubble or net attacking (exploitation phase), and the search for prey (exploration phase), behaviour of the humpback whales [167].

### 5.10.1 Encircling prey

Humpback whale recognizes the location of their prey and encircles them. The WOA starts with an initial population of solutions. The current best candidate solution is assume as the target prey. After the best search agent is defined, the other search agents will hence try to update their positions towards the best search agent which is mathematically formulated in equations 5.11 and 5.12.

\[
\begin{align*}
\vec{D} &= |\vec{C} \vec{X}^* - \vec{X}(t)| \quad 5.11 \\
\vec{X}(t + 1) &= \vec{X}^*(t) - \vec{A}\vec{D} \quad 5.12
\end{align*}
\]

where *t* indicates the current iteration, and \(\vec{A}\) and \(\vec{C}\) coefficient vectors, \(\vec{X}\) is the position vector and \(\vec{X}^*\) is the position vector of the best solution obtained so far. The vectors \(\vec{A}\) and \(\vec{C}\) are given by

\[
\begin{align*}
\vec{A} &= 2\vec{a} \cdot \vec{r} - \vec{a} \quad 5.13 \\
\vec{C} &= 2 \cdot \vec{r} \quad 5.14
\end{align*}
\]

where \(\vec{a}\) is linearly decreased from 2 to 0 over the course of iterations (in both exploration and exploitation phases) and \(\vec{r}\) is a random vector in [0,1].
5.10.2 Bubble-net attacking method

Humpback whales simultaneously swim around their prey within a shrinking circle and also along a spiral-shaped path, which is modelled with a 50% probability for updating the positions of the whales, either between the shrinking encircling mechanism or the spiral model as in equation 5.15.

\[ \mathbf{\tilde{X}}(t + 1) = \begin{cases} \mathbf{-A} \cdot \mathbf{D} & \text{if } p < 0.5 \\ \mathbf{D}^t \cdot e^{bi} \cdot \cos(2\pi l) + \mathbf{\bar{X}}^*(t) & \text{if } p \geq 0.5 \end{cases} \]  

5.15

To model the shrinking encircling mechanism the values of \( \mathbf{A} \) is chosen to be a random value in the interval \([-a,a]\), where \( a \) is decreased from 2 to 0 over the course of iterations. For modeling the spiral position update, a spiral equation is created between the position of whale and prey to mimic the helix shaped movement. \( \mathbf{D}^t = |\mathbf{\bar{X}}^*(t) - \mathbf{\tilde{X}}(t)| \) indicates the distance of the \( i^{th} \) whale to the best solution obtained so far, which indicates the position of the prey, \( b \) is a constant for defining the shape of the spiral and \( l \) is a random number in \([-1,1]\).

5.10.3 Search for prey

Humpback whales search randomly for prey. Hence, in order to model this random behaviour and allow exploration of the search space, the position of a randomly chosen search agent is updated instead of the best agent found so far. The mathematical model is as follows

\[ \mathbf{\bar{D}} = |\mathbf{C}\mathbf{\bar{X}}_{\text{rand}} - \mathbf{\bar{X}}(t)| \]  

5.16

\[ \mathbf{\tilde{X}}(t + 1) = \mathbf{\bar{X}}_{\text{rand}} - \mathbf{A}\mathbf{\bar{D}} \]  

5.17

To ensure exploitation, \( \mathbf{A} \) has random values either greater than 1 or less than -1.
Chapter 5 Classifier Optimisation

The pseudo-code of WOA is shown in Figure 5.8. The WOA algorithm starts with a population of random solutions. At each iteration, the search agents update their position with respect to the position of the randomly chosen search agent in the searching for prey phase, and with respect to the position of the best agent in the encircling phase.

```
Initialize the whale population, X_i, i = 1, 2, ...., n
Select the best search agent X*
while (t<maximum no. of iterations)
{
  update a, A, C, l and p for each search agent
  if (p<0.5)
    if(|A|<1)
      Update the position of the current search agent as in encircling phase
    else if(|A|≥1)
      Select a random search agent (X_rand)
      Update the position of the current search agent as in exploration phase
  end if
  else if(p≥0.5)
    Update the position of the current search agent as in exploitation (bubble net) phase
  endif
  Calculate fitness of each search agent
  Update X* if there is a better solution
  t=t+1;
}
```

Fig.5.8 WOA pseudo-code

The algorithm also chooses between shrinking encircling mechanism and the spiral model according to a randomly drawn probability measure. The algorithm effectively employs both exploration and exploitation. Adaptive variation of the search vector $\vec{A}$ allows the WOA algorithm to smoothly
transit between exploration and exploitation: by decreasing $A$, some iterations are devoted to exploration ($|A| \geq 1$) and the rest is dedicated to exploitation ($|A| \leq 1$). The algorithm is terminated when the termination criterion is reached.

The proposed SVM based underwater classifier was optimised with WOA. The results of the experiments for different population sizes are tabulated in Table 5-3. Each run of the algorithm consists of 500 iterations, except when the algorithm is terminated when the average relative change in fitness value is stalled over 25 generations.

<table>
<thead>
<tr>
<th>Population Size</th>
<th>10</th>
<th>15</th>
<th>20</th>
<th>25</th>
</tr>
</thead>
<tbody>
<tr>
<td>F-score (class 1)</td>
<td>0.62</td>
<td>0.63</td>
<td>0.70</td>
<td>0.67</td>
</tr>
<tr>
<td>F-score (class 2)</td>
<td>0.73</td>
<td>0.73</td>
<td>0.75</td>
<td>0.75</td>
</tr>
<tr>
<td>F-score (class 3)</td>
<td>0.72</td>
<td>0.72</td>
<td>0.73</td>
<td>0.73</td>
</tr>
<tr>
<td>F-score (class 4)</td>
<td>0.80</td>
<td>0.81</td>
<td>0.82</td>
<td>0.82</td>
</tr>
<tr>
<td>F-score (class 5)</td>
<td>0.83</td>
<td>0.85</td>
<td>0.86</td>
<td>0.86</td>
</tr>
<tr>
<td>F-score (class 6)</td>
<td>0.81</td>
<td>0.81</td>
<td>0.82</td>
<td>0.82</td>
</tr>
<tr>
<td>F-score (class 7)</td>
<td>0.73</td>
<td>0.74</td>
<td>0.75</td>
<td>0.75</td>
</tr>
<tr>
<td>F-score (class 8)</td>
<td>0.79</td>
<td>0.83</td>
<td>0.83</td>
<td>0.83</td>
</tr>
<tr>
<td>F-score (class 9)</td>
<td>0.76</td>
<td>0.80</td>
<td>0.81</td>
<td>0.81</td>
</tr>
<tr>
<td>F-score (class 10)</td>
<td>0.63</td>
<td>0.63</td>
<td>0.65</td>
<td>0.65</td>
</tr>
<tr>
<td>F-score (class 11)</td>
<td>0.61</td>
<td>0.62</td>
<td>0.67</td>
<td>0.66</td>
</tr>
<tr>
<td>F-score (Average)</td>
<td>0.73</td>
<td>0.74</td>
<td>0.76</td>
<td>0.76</td>
</tr>
<tr>
<td>Overall Accuracy (%)</td>
<td>68.7</td>
<td>70.2</td>
<td>73.1</td>
<td>72.0</td>
</tr>
</tbody>
</table>
5.11 Stochastic Fractal Search (SFS)

Stochastic Fractal Search (SFS) algorithm is a meta-heuristic algorithm, formulated by Salimi [171] based on the random fractals observed in nature. Fractal refers to objects or quantities that displays self similarity, in somewhat technical sense, in all dimensions [172]. Fractal shapes can be generated by common techniques such as iterated function systems, strange attractors, L-systems, escape time fractals, finite subdivision rules and random fractals. Random fractals can be created by physically motivated models such as diffusion limited aggregation (DLA) model. SFS algorithm is inspired from random fractals grown by DLA model, which can model clusters describing a bacterial colony. In DLA model, virtual particles moving through space following a random walk diffuse and stick together around a seed particle. A cluster is built up over time as more and more particles collide and clump together. While forming the cluster, the probability of the particle which sticks to the farthest end of the cluster is high in comparison to the one that penetrates the interior.

The problem solutions to an optimization problem are considered as individual particles in a population. SFS relies on three simple principles to find a solution

i. Each particle has an electrical potential energy which is updated based on its fitness value

ii. Each particle diffuses, and cause some other random particles to be created, and the energy of the seed particle is divided among generated particles

iii. The best generated particles from the diffusion process are considered, and the rest of the particles are eliminated
Two main processes to perform the SFS, are the diffusion process and the update process. In the diffusion process, the particles diffuse around its current position to form a cluster and in the updating process, the velocity and position of the particles is updated based on its position and the position of other particles.

The algorithm begins by randomly placing the particles at different locations in the search space. Each particle is initialized with equal energy obtained from equation 5.18.

\[ E_i = \frac{E}{P} \]  

5.18

where \( E \) is the maximum electrical potential energy considered to solve the problem.

After initialization, the fitness function of all particles is calculated and the best point \( BP \) is tracked. Each particle is then diffused in each generation, which creates new particles. The diffusion process is modelled through Gaussian walks as in equations 5.19 and 5.20.

\[ GW_1 = \text{Gaussian} \left( \mu_{BP}, \sigma \right) = (\varepsilon \cdot BP - \varepsilon' \cdot P_i) \]  

5.19

\[ GW_2 = \text{Gaussian} \left( \mu_P, \sigma \right) \]  

5.20

where \( \varepsilon \) and \( \varepsilon' \) are uniformly distributed random numbers in the range \([0,1]\). \( P_i \) is the \( i^{th} \) point, and \( BP \) is the best point in the cluster. \( \mu_{BP} \), \( \mu_P \) and \( \sigma \) are Gaussian parameters where \( \mu_{BP} \) is exactly equal to \( BP \) and \( \mu_P \) is equal to \( P \). For a particular generation \( g \), the standard deviation is computed as in equation 5.21.

\[ \sigma = \left[ \frac{\log(g)}{g} \cdot (P_i - BP) \right] \]  

5.21
The term \( \frac{\log(g)}{g} \) causes the size of Gaussian jumps to decrease as generations proceed, thereby forcing the algorithm to move closer to the solution. The diffusion process introduces the exploitation property of the algorithm and increases the chances of the algorithm in finding the optimal solutions.

After the diffusion process, all points (the total number being \( N \)) are ranked based on their individual fitness values, and each of them are assigned the probability of entering the next generation, the value of which follows a uniform distribution as in equation 5.22.

\[
P_{a_i} = \frac{\text{rank}(P_i)}{N} \tag{5.22}
\]

As per the above equation 5.22, the individual with a higher rank will have a higher probability to be selected into the next generation.

For each point \( P_i \) in the cluster, with \( P_{a_i} < \epsilon \), where \( \epsilon \) is a random value in the range \([0,1]\), the \( j^{th} \) component of \( P_i \) is updated using the equation 5.23,

\[
P'_i(j) = P_r(j) - \epsilon \cdot (P_{\ell}(j) - P_t(j)) \tag{5.23}
\]

where \( P'_i \) is the updated position of \( P_i \), and \( P_r \) and \( P_{\ell} \) are randomly selected points in the group. All the points are again sorted based on their ranks calculated by equation 5.22. Again the position of points with \( P_{a_i} < \epsilon \), where \( \epsilon \) is a random value in the range \([0,1]\), is updated according to equations 5.24 and 5.25.

\[
P''_i = P'_i - \bar{\epsilon} \cdot (P'_t - BP) \text{ for } \epsilon' \leq 0.5 \tag{5.24}
\]

\[
P''_i = P'_i + \bar{\epsilon} \cdot (P'_t - BP) \text{ for } \epsilon' > 0.5 \tag{5.25}
\]
where, \( P_l \) and \( P_t \) are randomly selected points and \( \hat{e} \) are random numbers generated by the Gaussian normal distribution. The new point \( P_l^m \) is replaced by \( P_l^i \) if its fitness function value is better than \( P_l^i \). The energy is distributed among the new particles according to their fitness value. The distribution energy equation for the selected particles is given by

\[
E_i^j = \left( \frac{f_j}{f_i + \sum_{k=1}^{n} f_k} \right) \times E_i
\]

where \( f_j \) is the energy of the diffused particle and \( f_i \) is its fitness value before diffusion and \( E_i \) is calculated as per equation 5.26. The updating process contributes to the exploration property of the SFS algorithm.

The proposed SVM based underwater classifier was optimised with SOS algorithm. The results of the experiments for different population sizes are tabulated in Table 5-4. Each run of the algorithm consists of 500 iterations, except when the algorithm is terminated when the average relative change in fitness value is stalled over 25 generations.

Table 5-4 Performance results with SFS algorithm based optimisation for varying population size

<table>
<thead>
<tr>
<th>Population Size</th>
<th>10</th>
<th>15</th>
<th>20</th>
<th>25</th>
</tr>
</thead>
<tbody>
<tr>
<td>F-score (class 1)</td>
<td>0.64</td>
<td>0.67</td>
<td>0.69</td>
<td>0.68</td>
</tr>
<tr>
<td>F-score (class 2)</td>
<td>0.67</td>
<td>0.69</td>
<td>0.70</td>
<td>0.69</td>
</tr>
<tr>
<td>F-score (class 3)</td>
<td>0.76</td>
<td>0.76</td>
<td>0.76</td>
<td>0.76</td>
</tr>
<tr>
<td>F-score (class 4)</td>
<td>0.90</td>
<td>0.91</td>
<td>0.93</td>
<td>0.91</td>
</tr>
<tr>
<td>F-score (class 5)</td>
<td>0.80</td>
<td>0.81</td>
<td>0.81</td>
<td>0.80</td>
</tr>
<tr>
<td>F-score (class 6)</td>
<td>0.81</td>
<td>0.82</td>
<td>0.82</td>
<td>0.82</td>
</tr>
<tr>
<td>F-score (class 7)</td>
<td>0.75</td>
<td>0.77</td>
<td>0.77</td>
<td>0.76</td>
</tr>
<tr>
<td>F-score (class 8)</td>
<td>0.79</td>
<td>0.80</td>
<td>0.80</td>
<td>0.79</td>
</tr>
</tbody>
</table>
5.12 Symbiotic Organisms Search

Symbiosis is derived from two Greek words ‘sym’, which means ‘together’ and ‘bios’ which means ‘life’ and can be stated as close and often long term interaction and reliance between two variant biological species. Symbiotic relationships exist in organisms as a strategy for them to adapt with changes in environment, which may help them increase their fitness and survival advantage in the eco-system over a long term. Symbiotic relationships can be obligate or facultative. Obligate symbiosis is when two organisms are in a symbiotic relationship because they can’t survive without each other. Facultative symbiosis is when the species live together by choice. Pollination symbiosis is an obligate symbiosis whereas cleaning symbiosis is a facultative symbiosis. There are three main types of symbiotic relationships: mutualism, commensalism and parasitism depending on the nature and character of interaction between the associated organisms. Cheng and Prayogo formulated Symbiotic organisms Search (SOS) algorithm based on symbiotic interaction strategies observed in nature [173]. Three phases are introduced in the SOS algorithm to resemble the biological interaction occurring in the three types of symbiosis.

SOS algorithm commences with a randomly generated initial population representing the initial ecosystem, in which each organism
corresponds to a candidate solution with an associated fitness value that reflects its survival cost in the ecosystem. In the initial ecosystem, a group of organisms are randomly generated within the search space. Each organism interacts with the other organisms randomly through all phases. This process is repeated for all the organisms in the ecosystems, until termination criteria are met. The pseudo-code of SOS algorithm is shown in Figure 5.9.

```
Initialize Ecosystem: define the no. of organisms
Define termination criteria
Calculate the fitness and find the initial best solution
while (termination criteria not met)
{
    for i = 1:ecosize
        Mutualism Phase
            Select one organism randomly, $X_i$ where $X_j \neq X_i$
            Determine Mutual Vector = $(X_i + X_j)/2$
            Determine Benefit Factors BF1 & BF2 = 1 or 2
            Modify $X_i$ and $X_j$ according to equations 5.27 & 5.28
            Calculate fitness values of modified organisms
            if(modified organisms are fitter than previous)
                accept modified organism to replace the previous
            else
                reject modified organisms and keep the previous
        Commensalism Phase
            Select one organism randomly, $X_i$ where $X_j \neq X_i$
            Modify organism $X_i$ according to equation 5.30
        Parasitism Phase
            Select one organism randomly, $X_i$ where $X_j \neq X_i$
            Create a Parasite Vector from Organism $X_i$
            Calculate fitness values of new organisms
            if(Parasite Vector fitter than $X_j$)
                replace organism $X_j$ with Parasite Vector
            else
                Keep organism $X_j$ and delete Parasite Vector
        }
```

Fig.5.9 SOS pseudo-code

5.12.1 Mutualism Phase

Mutualism is a symbiotic relationship in which each individual benefit from the activity of the other, like the relationship between bees and flowers
in which the bee benefits from the nectar it gets from the flower and the flower benefits from the pollination carried out by the bees.

Mutualistic relation can be mathematically modelled as follows. An organism \( X_i \) engages in a mutualistic relationship with another organism \( X_j \) with the goal of increasing their mutual survival advantage in the ecosystem. Based on the mutualistic symbiosis between organisms \( X_i \) and \( X_j \), the new candidate solutions are calculated as follows

\[
\begin{align*}
X_{i_{\text{new}}} &= X_i + \text{rand}(0,1)(X_{\text{best}} - \text{Mutual Vector} \times BF_1) \quad 5.27 \\
X_{j_{\text{new}}} &= X_j + \text{rand}(0,1)(X_{\text{best}} - \text{Mutual Vector} \times BF_2) \quad 5.28 \\
\text{Mutual Vector} &= \frac{X_i + X_j}{2} \quad 5.29
\end{align*}
\]

\( BF_1 \) and \( BF_2 \) are factors which determine the degree of benefit to each organism.

This phase aids in exploration of new regions, as organisms located far away in the search space are brought to interact by \textit{Mutual Vector}. Further, the two interacting individuals are updated concurrently rather than singly.

5.12.2 Commensalism Phase

Commensalism is a symbiotic relationship in which one organism (the commensal) benefits, and the other is apparently unaffected (or receives minimal benefit) like that of algae and barnacles growing on turtles and whales. The new candidate solutions in this phase are found by exploiting promising regions around the best solution. This phase controls the convergence of the algorithm and is mathematically modelled as follows

\[
\begin{align*}
X_{i_{\text{new}}} &= X_i + \text{rand}(-1,1)(X_{\text{best}} - X_j) \quad 5.30
\end{align*}
\]
Parasitism Phase

Parasitism is a symbiotic relationship in which one organism (the parasite) benefits, at the expense of other such that the other organism is adversely affected, like fleas harming the hosts on which they live. To model this phase a Parasite Vector is created in the search space by duplicating organism $X_i$, and modifying it along a randomly selected dimension. Both the organisms are then evaluated, and the fitter organism is allowed to resume its position in the ecosystem. Parasitism phase allows exploration of the search space and may arrive at unique solutions that may be located in completely different regions of the search space.

The proposed SVM based underwater classifier was optimised with SOS algorithm. The results of the experiments for different population sizes are tabulated in Table 5-5. Each run of the algorithm consists of 500 iterations, except when the algorithm is terminated when the average relative change in fitness value is stalled over 25 generations.

<table>
<thead>
<tr>
<th>Population Size</th>
<th>10</th>
<th>15</th>
<th>20</th>
<th>25</th>
</tr>
</thead>
<tbody>
<tr>
<td>F-score (class 1)</td>
<td>0.68</td>
<td>0.75</td>
<td>0.71</td>
<td>0.69</td>
</tr>
<tr>
<td>F-score (class 2)</td>
<td>0.76</td>
<td>0.78</td>
<td>0.78</td>
<td>0.76</td>
</tr>
<tr>
<td>F-score (class 3)</td>
<td>0.74</td>
<td>0.76</td>
<td>0.75</td>
<td>0.74</td>
</tr>
<tr>
<td>F-score (class 4)</td>
<td>0.88</td>
<td>0.89</td>
<td>0.89</td>
<td>0.88</td>
</tr>
<tr>
<td>F-score (class 5)</td>
<td>0.90</td>
<td>0.92</td>
<td>0.91</td>
<td>0.90</td>
</tr>
<tr>
<td>F-score (class 6)</td>
<td>0.78</td>
<td>0.79</td>
<td>0.78</td>
<td>0.78</td>
</tr>
<tr>
<td>F-score (class 7)</td>
<td>0.88</td>
<td>0.89</td>
<td>0.88</td>
<td>0.88</td>
</tr>
<tr>
<td>F-score (class 8)</td>
<td>0.85</td>
<td>0.86</td>
<td>0.85</td>
<td>0.85</td>
</tr>
</tbody>
</table>
5.13 Improving the Parameter Optimisation by Modified - Symbiotic Organisms Search

The right balance between a meta-heuristic algorithm’s ability to explore and exploit determines its efficiency to arrive at the optimal solution. Exploitation ability is introduced in meta–heuristic algorithms usually by directing the search towards the best solution. In the proposed modified symbiotic organisms search (m-SOS) algorithm, a balanced exploitation is introduced through a cognitive component as well as a social component. The cognitive component is dependent on the best value of a particular organism ($X_{best}$) and the social component is dependent on the best value among all the organisms ($G_{best}$). A better coverage of the search space is achieved through the social component and the cognitive component directs the search towards the possibly best solution in the neighbourhood. The social component reduces the algorithms chance to get trapped in a local minima. In the proposed m–SOS algorithm another modification proposed is to introduce a weighted mutual vector. By introducing so, the factor $Mutual Vector \times BF$ in the SOS algorithm controls the degree by which the candidate solution approaches the best solution. The weights are determined by the fitness values obtained in the previous iterations. Hence, it may be considered as the momentum component which determines the impetus with which the candidate solution moves towards the best solution, based on the

<table>
<thead>
<tr>
<th>F-score (class 9)</th>
<th>0.85</th>
<th>0.87</th>
<th>0.87</th>
<th>0.85</th>
</tr>
</thead>
<tbody>
<tr>
<td>F-score (class 10)</td>
<td>0.71</td>
<td>0.73</td>
<td>0.73</td>
<td>0.71</td>
</tr>
<tr>
<td>F-score (class 11)</td>
<td>0.69</td>
<td>0.73</td>
<td>0.70</td>
<td>0.69</td>
</tr>
<tr>
<td>F-score (Average)</td>
<td>0.79</td>
<td>0.82</td>
<td>0.80</td>
<td>0.79</td>
</tr>
<tr>
<td>Overall Accuracy (%)</td>
<td>77.5</td>
<td>80.4</td>
<td>78.9</td>
<td>76.7</td>
</tr>
</tbody>
</table>
previous experience in terms of the fitness value. The right balance between exploration and exploitation determines the efficacy of the algorithm. In m–SOS algorithm exploration is also introduced through the GA operators crossover and mutation. Randomly generated indices of the solutions $X_{i\text{new}}$ and $X_{j\text{new}}$ are subjected to crossover operators to generate new solutions $X'_{i\text{new}}$ and $X'_{j\text{new}}$. The resulting solutions $X'_{i\text{new}}$ and $X'_{j\text{new}}$ if found to have better fitness values are accepted, else $X_{i\text{new}}$ and $X_{j\text{new}}$ are retained. In the parasitism phase, the organism is randomly mutated to generate the parasite vector. The algorithm can be described in detail as follows

### 5.13.1 Mutualism Phase

The new candidate solutions for $X_i$ and $X_j$ are calculated based on the mutualistic symbiosis between organism $X_i$ and $X_j$.

\[
X_{i\text{new}} = X_i + \text{rand}(0,1) (X_{best} - X_i) - \text{Mutual Vector} \times BF_1 (G_{best} - X_i) \quad 5.31
\]

\[
X_{j\text{new}} = X_j + \text{rand}(0,1) (X_{best} - X_j) - \text{Mutual Vector} \times BF_2 (G_{best} - X_j) \quad 5.32
\]

$X_{best}$ is the best value of that particular organism, and $G_{best}$ is the best value among all organisms.

$BF_1$ and $BF_2$ are factors which determine the degree of benefit to each organism

\[
\text{Mutual Vector} = \left(w_1 X_i + w_2 X_j\right) / 2 \quad 5.33
\]

where $w_1 = \eta_i - 1 / 100$ and $w_2 = \eta_j - 1 / 100$

$\eta_i - 1$ is the percentage of correctly classified samples in the previous iteration
Chapter 5 Classifier Optimisation

\[ \text{Mutual Vector} = \left( \frac{w_1 X_i + w_2 X_j}{2} \right) \]

where \( w_1 = \eta_{i-1}/100 \) and \( w_2 = \eta_{j-1}/100 \)

\[ \text{id}x = [i_1, i_2, \ldots, i_M], M < N \]

\( i_1, i_2, \ldots, i_M \) are random numbers between 1\&N

\[ X_{CO} \in X_i \quad \forall x \in X_{CO} \rightarrow x \in X_i \]

\[ X_{CO}(i) = X_i(idx(i)), i = 1, \ldots, M \]

\[ Y_{CO} \in X_j \quad \forall y \in Y_{CO} \rightarrow y \in Y_i \]

\[ Y_{CO}(i) = X_j(idx(i)), i = 1, \ldots, M \]

\[ X' = X_i(idx(i)) = Y_{CO}(i) \]

\[ Y' = X_j(idx(i)) = X_{CO}(i) \]

The new solutions can be calculated as

\[ X'_{inew} = X' + \text{rand}(0,1)(X_{best} - X_i) \]

\[ X'_{jnew} = Y' + \text{rand}(0,1)(X_{best} - X_j) \]

accept the best solutions

\[ \text{if} \ (\text{fitness} (X'_{inew}) > \text{fitness}(X_{inew})) \]

accept \( X'_{inew} \) else retain \( X_{inew} \)

\[ \text{if} \ (\text{fitness} (X'_{jnew}) > \text{fitness}(X_{jnew})) \]

accept \( X'_{jnew} \) else retain \( X_{jnew} \)
5.13.2 Commensalism Phase

The new candidate solutions in this phase are calculated as

$$X_{inew} = X_i + rand(-1,1)(X_{best} - X_j)(G_{best} - X_j)$$  \hspace{1cm} (5.41)

The proposed SVM based underwater classifier was optimised with \textit{m}-SOS algorithm. The results of the experiments for different population sizes are tabulated in Table 5.6. Each run of the algorithm consists of 500 iterations, except when the algorithm is terminated when the average relative change in fitness value is stalled over 25 generations.

Table 5.6 Performance results with the proposed \textit{m}-SOS algorithm based optimisation for varying population size

<table>
<thead>
<tr>
<th>Population Size</th>
<th>10</th>
<th>15</th>
<th>20</th>
<th>25</th>
</tr>
</thead>
<tbody>
<tr>
<td>F-score (class 1)</td>
<td>0.77</td>
<td>0.81</td>
<td>0.80</td>
<td>0.78</td>
</tr>
<tr>
<td>F-score (class 2)</td>
<td>0.81</td>
<td>0.84</td>
<td>0.84</td>
<td>0.82</td>
</tr>
<tr>
<td>F-score (class 3)</td>
<td>0.74</td>
<td>0.79</td>
<td>0.79</td>
<td>0.76</td>
</tr>
<tr>
<td>F-score (class 4)</td>
<td>0.90</td>
<td>0.91</td>
<td>0.91</td>
<td>0.90</td>
</tr>
<tr>
<td>F-score (class 5)</td>
<td>0.92</td>
<td>0.93</td>
<td>0.93</td>
<td>0.93</td>
</tr>
<tr>
<td>F-score (class 6)</td>
<td>0.71</td>
<td>0.76</td>
<td>0.76</td>
<td>0.75</td>
</tr>
<tr>
<td>F-score (class 7)</td>
<td>0.93</td>
<td>0.95</td>
<td>0.95</td>
<td>0.94</td>
</tr>
<tr>
<td>F-score (class 8)</td>
<td>0.85</td>
<td>0.87</td>
<td>0.86</td>
<td>0.86</td>
</tr>
<tr>
<td>F-score (class 9)</td>
<td>0.84</td>
<td>0.88</td>
<td>0.88</td>
<td>0.85</td>
</tr>
<tr>
<td>F-score (class 10)</td>
<td>0.78</td>
<td>0.82</td>
<td>0.82</td>
<td>0.80</td>
</tr>
<tr>
<td>F-score (class 11)</td>
<td>0.79</td>
<td>0.84</td>
<td>0.82</td>
<td>0.81</td>
</tr>
<tr>
<td>F-score (Average)</td>
<td>0.82</td>
<td>0.85</td>
<td>0.85</td>
<td>0.84</td>
</tr>
<tr>
<td>Overall Accuracy (%)</td>
<td>82.5</td>
<td>85.1</td>
<td>88.4</td>
<td>83.3</td>
</tr>
</tbody>
</table>
Chapter 5 Classifier Optimisation

5.14 Summary

This chapter throws light into optimisation of a classifier and the various concepts associated with it. Different optimisation strategies that can be adopted are briefed. Meta-heuristic optimisation algorithms strike a balance between exploration and exploitation of the search space. Meta-heuristic optimisation is adopted in this work for parameter optimisation. The various meta-heuristic optimisation algorithms adopted are described.

The results of adoption of optimisation algorithms to the proposed classifier, has improved its performance. Thus, it can be concluded that, tuning the algorithmic parameters to the optimum is essential for improving the classifier performance.