T.C
ERCIYES UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
DEPARTMENT OF MECHATRONICS ENGINEERING

ON THE OPTIMIZATION OF SUPPORT VECTOR MACHINES PERFORMANCE FOR WHEAT CLASSIFICATION

Prepared by
SHAWQI MOHAMMED OTHMAN FAREA

Supervisor
Assoc. Prof. Dr. Muzaffer KANAAN

MS Thesis

July 2020
KAYSERI

T.C
ERCIYES UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
DEPARTMENT OF MECHATRONICS ENGINEERING

ON THE OPTIMIZATION OF SUPPORT VECTOR MACHINES PERFORMANCE FOR WHEAT CLASSIFICATION

(MS Thesis)

Prepared by
SHAWQI MOHAMMED OTHMAN FAREA

Supervisor
Assoc. Prof. Dr. Muzaffer KANAAN

July 2020
KAYSERI

SCIENTIFIC ETHICS SUITABILITY

I, hereby, declare that all information in this work was obtained in accordance with the academic and ethical rules. All results and materials that have not been at the essence of this work are also transferred and expressed by giving reference as required by these rules and behavior.

Shawqi Mohammed Othman Farea

The MS thesis entitled **"On the Optimization of Support Vector Machines Performance for Wheat Classification"** has been prepared in accordance with Erciyes University Graduate Education and Teaching Institute Thesis Preparation and Writing Guide.

Student

Shawqi Mohammed Othman Farea

Supervisor

Assoc. Prof. Dr. Muzaffer KANAAN

Head of Mechatronics Engineering Department

Prof. Dr. Şahin YILDIRIM

This study entitled **"On the Optimization of Support Vector Machines Performance for Wheat Classification"** prepared by **"Shawqi Mohammed Othman Farea"** under the supervision of **Assoc. Prof. Dr. Muzaffer KANAAN** was accepted by the jury as MS Thesis in Mechatronics Engineering department.

….. /…… / 2020

JURY:

Supervisor : Assoc. Prof. Dr. Muzaffer KANAAN

Juror : Assist. Prof. Dr. Rüştü AKAY

Juror : Assist. Prof. Dr. Mehmet BİLİM

APPROVAL

That the acceptance of this thesis has been approved by the decision of the Institute's Board of Directors with the ……...……. date and ……. numbered decision.

……. /……../ ……

Prof. Dr. Mehmet AKKURT

Director of the Institute

ACKNOWLEDGEMENTS

First and foremost, I would like to heartily thank my supervisor Assoc. Prof. Dr. Muzaffer KANAAN for his unwavering support and helpful guidance throughout my master's degree. I deem myself fortunate to work under his supervision. I thank him sincerely for his advice, encouragement, and patience.

I would like also to thank Assist. Prof. Rüştü AKAY for his help and support during my master's degree. In addition, I would like to thank Res. Asst. Memduh SUVEREN and my dear friend Radhwan Saleh for their great help during the thesis work and preparation.

I would like to thank Türkiye Scholarships for giving me this opportunity to complete my master's degree in this great country: Turkey.

I am also grateful to Microsoft Azure for providing me, as a student, the chance to conduct some of my simulation experiments on their virtual machines.

Last but not least, I would like to express my special thanks to my great family, in particular my parents, for their endless support and help.

Shawqi Mohammed Othman Farea

KAYSERI, 2020

ON THE OPTIMIZATION OF SUPPORT VECTOR MACHINES PERFORMANCE FOR WHEAT CLASSIFICATION

SHAWQI MOHAMMED OTHMAN FAREA
Erciyes University, Graduate School of Natural and Applied Sciences
MS Thesis, July 2020
Supervisor: Assoc. Prof. Dr. Muzaffer KANAAN

ABSTRACT

Support vector machines (SVMs), renowned for their high generalization ability, are a widely used, powerful Machine Learning algorithm. Yet, they, like other learning algorithms, involve some hyper-parameters whose values play a crucial role in their performance and require to be tuned according to the dataset at hand. Consequently, this work aims primarily to improve the performance of four different types of SVMs by efficiently tuning their hyper-parameters. These four types are Gaussian, polynomial, sigmoid SVMs, and a weighted sum of Gaussian and polynomial SVMs (hereafter known as hybrid-kernel SVMs). Five meta-heuristic optimization algorithms; namely artificial bee colony (ABC), teaching-learning-based optimization (TLBO) algorithm, cuckoo search (CS), ABC with the depth-first-search framework (DFSABC), and teaching-learning-based ABC (TLABC); are implemented to tune the hyper-parameters according to the wheat varieties dataset. They search for the optimum hyper-parameters leading to the minimum five-fold cross-validation error. The results demonstrate that the performance of the four SVM classifiers is improved significantly, such that the best classification accuracies of Gaussian, polynomial, sigmoid and hybrid-kernel SVMs are 98.57%, 98.10%, 98.10% and 97.62% respectively. These results outperform those found in the literature. Overall, Gaussian SVMs rank first while sigmoid SVMs rank second with respect to the classification accuracy and computational time combined. As regards the meta-heuristic algorithms, ABC has the worst convergence in this problem, whereas the other four algorithms exhibit almost similar convergences. Nonetheless, DFSABC is a bit better than the others when both convergence and computational time are considered.

Keywords: SVMs, Hyper-parameters, Meta-heuristic algorithms, Wheat varieties, Classification, Statistical analysis

DESTEK VEKTÖR MAKİNELERİNİN PERFORMANSININ BUĞDAY SINIFLANDIRMASI İLE OPTİMİZASYONU HAKKINDA

SHAWQI MOHAMMED OTHMAN FAREA
Erciyes Üniversitesi, Fen Bilimleri Enstitüsü
Yüksek Lisans Tezi, Temmuz 2020
Danışman: Doç. Dr. Muzaffer KANAAN

ÖZET

Yüksek genelleme yetenekleriyle tanınan destek vektör makineleri (SVM'ler) yaygın olarak kullanılan güçlü bir makine öğrenme (machine learning) algoritmasıdır. Bununla birlikte, değerleri performanslarında önemli bir rol oynayan ve eldeki veri kümesine göre ayarlanması gereken bazı hiper-parametreleri içerirler. Sonuç olarak, bu çalışma öncelikle hiper-parametrelerini etkili bir şekilde ayarlayarak dört farklı SVM tipinin performansını artırmayı amaçlamaktadır. Bu dört tür, Gauss, polinom, sigmoid SVM'ler ve ağırlıklı bir Gauss ve polinom SVM toplamıdır (bundan böyle hybrid-kernel SVM'ler olarak anılacaktır). Bu çalışma kapsamında, buğday çeşitlerini sınıflandırmak amacı ile veri kümesine göre hiper-parametreleri ayarlamak için beş meta-sezgisel optimizasyon algoritması uygulanmakta ve yukarıda anlatılan SVM performansları Buğday Çeşitleri very kümesine (wheat varieties data set) uygulanmaktadır. Bu beş algoritma, yapay arı kolonisi (ABC), öğretme-öğrenme tabanlı optimizasyon (TLBO) algoritması, guguklu arama (CS), derinlemesine arama çerçevesine sahip ABC (DFSABC) ve öğretme-öğrenme tabanlı ABC'dir (TLABC). Sonuçlar, dört SVM sınıflandırıcısının performansının önemli ölçüde iyileştirildiğini göstermektedir. Gauss, polinom, sigmoid ve hybrid-kernel SVM'lerin en iyi sınıflandırma doğruluğu sırasıyla %98.57, %98.10, %98.10 ve %97.62'dir. Bu sonuçlar literatürde, aynı veri kümesi baz alınarak daha önce belgelenmiş sonuçlardan daha iyidir. Genel olarak, Gauss SVM'leri birinci, sigmoid SVM'ler ise sınıflandırma doğruluğu ve hesaplama süresine göre ikinci sırada yer almaktadır. Meta-sezgisel algoritmalar ile ilgili olarak, ABC bu problemde en kötü yakınsamaya sahipken, diğer dört algoritma neredeyse benzer yakınsamalar sergiler. Bununla birlikte, hem yakınsama hem de hesap zamanı göz önüne alındığında, DFSABC diğerlerinden biraz daha iyi performans sergilediği tespit edilmiştir.

Anahtar Kelimeler: SVM'ler, Hiper-parametreleri, Meta-sezgisel algoritmalar, Buğday çeşitleri, sınıflandırma, istatistiksel analiz

TABLE OF CONTENTS

ON THE OPTIMIZATION OF SUPPORT VECTOR MACHINES PERFORMANCE FOR WHEAT CLASSIFICATION

CHAPTER 3
META-HEURISTIC OPTIMIZATION ALGORITHMS

CHAPTER 4
METHODOLOGY

CHAPTER 5
EXPERIMENTAL RESULTS AND DISCUSSION

LIST OF ABBREVIATIONS

Acronym	Full Name
ABC	Artificial Bee Colony
ACO	Ant Colony Optimization
ANFIS	Adaptive Neuro-Fuzzy Inference System
ANN	Artificial Neural Network
CS	Cuckoo Search
DBTC	The Distance Between Two Classes Index
DFSABC	ABC With Depth-First Search Framework
ECOC	Error-Correcting Output Codes
ERM	Empirical Risk Minimization
ESDR	The Expected Square Distance Ratio
GA	Genetic Algorithm
LSSVM	Least Square Support Vector Machine
ML	Machine Learning
MLP	Multi-Layer Perceptron
PCA	Principal Component Analysis
PNN	Probabilistic Neural Network
PSO	Particle Swarm Intelligence
RBF	Radial Basis Function
SMO	Sequential Minimal Optimization
SRM	Structural Risk Minimization
SVMs	Support Vector Machines
TLABC	Teaching-Learning-Based ABC
TLBO	Teaching-Learning-Based Optimization

LIST OF TABLES

LIST OF FIGURES

INTRODUCTION

SVMs, also known as kernel machines, are among the most commonly used machine learning (ML) algorithms. Indeed, SVMs are a very powerful and effective learning algorithm. Form a statistical perspective, SVMs are nonparametric algorithms where a priori knowledge of the data distribution is not required [1]. Similar to their key ML counterpart, artificial neural networks (ANNs), SVMs are a supervised learning technique mostly used for either regression (also known as function approximation) or classification (also known as pattern recognition). Unlike ANNs, which mainly try to minimize the empirical error, the SVM algorithm tries instead to minimize an upper bound of the generalization error. In fact, the so-called generalization error (as known as the true or expected error) measures the ability of a learning algorithm to accurately predict unseen test data. This type of error is often impossible to compute because it depends on the underlying data distribution, which is often unknown. However, an estimate or an upper bound of the generalization error is computable. More specifically speaking, the so-called empirical error, which is another name for the training error, is an estimate of the true error, but it is a good, unbiased estimate only when the data is big enough. On the contrary, minimizing an upper bound rather than minimizing the empirical error ensures small generalization errors regardless of the data size. Accordingly, SVMs show good generalization performance [2].

Like other learning algorithms, SVMs have extra hyper-parameters in addition to their normal model-defining parameters (i.e., coefficients of the hyperplane separating the data classes) that the algorithm attempts to find during the training process. Hyper-parameters, in fact, are parameters that have to be specified for an ML algorithm prior to the execution of the training process. The values of these parameters significantly influence the performance of SVMs. Unfortunately, there are no universal values for these hyper-

parameters applying to any dataset and any problem; instead, these hyper-parameters must be tuned according to the problem at hand using a particular optimization scheme. In the literature, the process of tuning the hyper-parameters of learning algorithms is known as *model selection*.

As regards SVMs, the main hyper-parameters are the regularization-parameter-like trade-off parameter C (also known as the penalty factor) as well as the kernel parameters. While the kernel parameters relate to the kernel function, the trade-off parameter plays a regularization role by controlling the balance between the training error (underfitting problem) and algorithm complexity (overfitting problem).

Accordingly, this study will focus on the optimization of the SVM hyper-parameters (the trade-off parameter and kernel parameters) using and comparing four different choices of the kernels: the Gaussian kernel, polynomial kernel, sigmoid kernel and a hybrid kernel, which is a weighted sum of both of the Gaussian kernel and the polynomial kernel. As the objective function of the optimization problem, the five-fold cross-validation is used, which, in fact, is an appropriate objective function for our benchmark dataset. The five-fold cross-validation scheme is more accurate and suitable than the holdout testing scheme (i.e., holding out a particular portion of the data for testing) since the dataset is small, with only 210 observations. Regarding the minimization algorithm, this study uses five different meta-heuristic algorithms, namely ABC [3,4], TLABC [5], DFSABC [6], TLBO [7], and lastly CS [8,9]. This framework is applied to the wheat varieties dataset, and the results will be compared with those of the previous studies. The major aim of this work is to improve the classification accuracy of the SVMs with the wheat varieties dataset while maintaining a reasonable level of time consumption and computational complexity.

The wheat varieties dataset is relatively small yet complicated, with three overlapped classes and seven attributes (i.e., features). It contains three classes of wheat varieties, namely Canadian, Kama and Rosa, each of which comprises 70 images or observations (more details about the dataset are discussed in Section 4.2) [10]. This dataset is comprised of X-ray images of wheat kernels. In fact, X-ray photography is a non-destructive approach for discriminating and classifying the wheat varieties.

To sum up, the main theme of this study is to enhance the performance of SVMs by means of the above-mentioned framework. The unique contributions of this study can be summarized as follows:

1. Optimizing kernel functions other than the Gaussian kernel, namely the sigmoid kernel, polynomial kernel and a hybrid kernel
2. Applying two new meta-heuristic optimization algorithms, namely TLABC and DFSABC, to the problem of SVM hyper-parameters optimization (To the best of our knowledge, no one before has applied TLABC or DFSABC to the optimization of the SVM hyper-parameters). Both algorithms have been proposed recently, and both showed promising results in many benchmark objective functions.
3. Using five-fold cross-validation together with the SVM algorithm in the wheat varieties dataset
4. Using different SVM kernels with the dataset of wheat varieties.

Finally, the importance of this work should be noticed through the considerable improvement in the classification accuracy of the wheat varieties dataset. Also, the results obtained by optimizing SVMs by the above-proposed methodology are remarkably better than those which already exist in the literature.

This thesis is divided and organized into six chapters. Chapter 1 presents a literature review about the previous works related to the wheat varieties dataset and the SVM hyper-parameters. Chapter 2 introduces SVMs and the detailed mathematical foundation of this algorithm. Chapter 3 presents the five meta-heuristic algorithms that are used in this work. Chapter 4 explains the herein proposed methodology in detail. Chapter 5 shows the experimental results and holds some discussion about these results. Lastly, Chapter 6 concludes this thesis by summarizing the main points and proposing some directions for possible future study

CHAPTER 1

REVIEW OF PRIOR LITERATURE

Due to its generalization ability, SVMs have been used with an enormous number of benchmark datasets and real-world problems across multiple fields. For instance, SVMs have been used in handwriting recognition [11–13], face and facial expression recognition [14–17], speech recognition [18,19], text detection and categorization [20,21], human activity recognition [22,23].

Moreover, the SVM algorithm has been applied to many agricultural applications. Some of these applications are plant diseases detection and classification [24–26], weed identification [27], soil type classification [28] detection of stored grain pests [29,30] and classification of stored grain status [31]. The following section sheds light on the last application, which is a preliminary work of this thesis.

1.1 Preliminary Work: SVM Classification of Stored Grain Condition

Monitoring the condition of the stored grain is imperative due to the importance of grain as a food source. In [32], a wireless network of moisture and temperature sensors was built and used to monitor the silobag-stored grain and to classify its status into three categories: safe, risky and dangerous. In that study, neural networks with three different training algorithms (i.e., scaled conjugate gradient, gradient descent with momentum as well as Levenberg-Marquardt backpropagation) were used to conduct this classification process based on the temperature degree, together with the moisture content, of the grain storage. The best classification accuracy reported in that study was roughly 99%.

As preliminary work of this thesis, SVMs were used so as to classify the condition of the silobag-stored grain as well [31]. The dataset collected in [32] was used, and the results

of this preliminary work were compared to those in [32]. The dataset consists of 300,000 data points, 100,000 for each class. As in [32], in this preliminary work, SVMs tried to classify the status of the grain stored in silo-bags into three distinct classes: safe, risky and dangerous. Owing to the multi-class classification nature of this problem, a couple of multi-class SVM algorithms were used with this problem. These types are one-versus-all and error-correcting output codes (ECOC) algorithms (more details about these two algorithms are presented in Section 2.5). The workflow in this preliminary work is divided into two stages:

1. **Preprocessing:** The data was first decreased from 300,000 examples into below 82,000 examples by eliminating any duplicates of the data points. Then, the data features (temperature and moisture) were normalized to fall into the [0,1] range. Also, the dataset was split into training and testing subsets. Eighty per cent of these data were used for training, while the remaining 20% were set aside as the testing set.
2. **Classification:** the one-versus-all SVM algorithm and ECOC SVM algorithm were implemented for classifying the preprocessed data.

The classification accuracy achieved in this work was 100% for both the one-versus-all and ECOC SVM algorithms. That means all the testing data were correctly classified, and, consequently, these results were better than those reported in [32]. Although both of the one-versus-all and ECOC algorithms achieved the 100% classification accuracy, the ECOC algorithm was almost 17 times faster than the one-versus-all algorithm using MATLAB.

The main work of this thesis, besides this preliminary work, is optimizing the SVM hyper-parameters, as explained in Section 1.3, and then classifying the wheat varieties dataset. The subsequent section introduces the wheat varieties dataset and conducts a brief literature review about prior work.

1.2 Wheat Varieties Dataset

In addition to the aforementioned agricultural applications, the kernel-based classification of wheat varieties is another application where SVMs, together with other ML algorithms, have been used to carry out the classification process. The wheat varieties dataset was

collected in [10] and uploaded to the UCI Machine Learning Repository [33], in which it is known as the seeds dataset. Then, a considerable number of succeeding studies have considered this dataset as a benchmark dataset ever since. A more detailed discussion on the structure of this dataset will be given in Section 4.2.

1.2.1 Related Work

In [10], the authors implemented the complete gradient clustering algorithm to the dataset and compared the obtained results to the K-means clustering algorithm. The average classification accuracy for both algorithms were 92%. In terms of the distinct classes, the classification accuracy was 84% for Kama whilst the accuracy for both Rosa and Canadian was 96% using the complete gradient clustering algorithm. On the other hand, the K-means clustering algorithm achieved a classification accuracy of 93% for Kama, 94% for Rosa and 89% for Canadian.

In [34], the least squares support vector machine (LSSVM), which is just a simplified version of the standard SVMs, was applied to multiple UCI Machine Learning datasets, one of which is the wheat varieties dataset. Three meta-heuristic algorithms; namely ABC, particle swarm optimization (PSO) and genetic algorithm (GA); were run to tune the hyper-parameters of LSSVM. The dataset was divided into two-thirds for training and one-third for testing. The average classification accuracy of LSSVM was 90% using GA, PSO and ABC as the hyperparameter-optimizer.

In [35], SVMs, ANNs and adaptive neuro-fuzzy inference system (ANFIS) were applied to this dataset as well. The algorithms were trained using 90% of the dataset, and they were tested by the remaining ten per cent. The reported classification accuracy in this study was 95.23% for SVMs and 100% for both ANNs and ANFIS.

In [36], four different learning algorithms were used for the classification of the wheat varieties dataset. These algorithms were multilayer perceptron algorithm (MLP), Naïve Bayes algorithm, K-nearest neighbor algorithm (KNN) and J48 decision tree algorithm. The classification accuracy reported in this work was 97.1% for MLP with seven neurons, 95.7% for KNN with four neighbors, 91.9% for J48 decision tree algorithm, and finally 91.4% for Naïve Bayes algorithm.

In [37], a Sammon mapping clustering algorithm was proposed to be combined with the intuitionistic fuzzy c-least squares support vector regression (IFC-LSSVR). This proposed combination was applied to the wheat varieties dataset. Also, a comparison between the results of this study and those of the fuzzy C-means and K-means clustering algorithms was drawn. The best classification accuracies for IFC-LSSVR, fuzzy C-means and K-means were 94.76%, 92.86% and 92.38%, respectively. That is, IFC-LSSVR outperformed the fuzzy C-means clustering as well as the K-means clustering.

In [38], probabilistic neural network (PNN) with local sensitivity analysis (LSA) was used with the wheat varieties dataset. As a measure of the classification accuracy, the 10-fold cross-validation was used in this study. The best achieved classification accuracy using PNN with LSA was approximately 93.4%.

Table 1.1 summarizes the previous relevant works that used different learning algorithms to classify the wheat varieties dataset. It shows the learning algorithms used in each study, together with the results reported in that study.

1.2.2 Limitations of SVM-related work

In total, three studies of the six above-mentioned studies used SVMs or their variants in the classification of the wheat varieties dataset. One important point worth noticing is that only one study of these three studies used the standard SVM algorithm whilst the other two used LSSVM (one simplified variant of the standard SVM). Here, a discussion of some limitations of these three studies is presented.

1. IFC-LSSVR was used as a clustering algorithm in [37]. Indeed, clustering is an unsupervised algorithm, which, unlike supervised algorithms, does not require any division of data into two separate subsets (i.e., training and testing/validation subsets). Moreover, clustering ignores completely our prior knowledge about the labels of the training data. Consequently, it is not appropriate for this dataset where it generally shows worse results, as compared with supervised algorithms. In contrast, the other two SVM-related studies [34,35] used SVMs and LSSVM as a supervised algorithm and divided the dataset into two subgroups, one for training and the other for testing. Nonetheless, the size of the dataset is small (only 210 examples); therefore, using cross-validation error leads to more accurate

results than the holdout testing error. In cross-validation, every data point is used for training and testing. In essence, cross-validation is time-consuming yet accurate, so it is recommended particularly for small datasets. On the other hand, holdout testing/validation is recommended especially for big datasets because it causes a reduction in the training data. To sum up, the cross-validation error is an unbiased estimate of the true error, yet it is computationally demanding and thus time-consuming. Therefore, it is preferred for small datasets. In contrast, the holdout testing/validation error is an unbiased estimate of the true error only when the dataset is big enough. Therefore, it is preferred for big datasets due to its good computational efficiency [39,40].

Table 1.1 Previous works related to the wheat varieties dataset

Paper	**Learning Algorithms**	**Classification Accuracy**
M. Charytanowicz, et al.[10]	Complete gradient clustering	92%
	K-means clustering	92%
P. Chen, et al.[34]	LSSVM with GA, PSO and ABC	90%
A. Kayabasi, et al.[35]	SVMs	95.23%
	ANNs	100%
	ANFIS	100%
K. Sabancı, et al.[36]	MLP	97.1%
	KNN	95.7%
	J48	91.9%
	Naïve Bayes	91.4%
K. Lin, et al.[37]	IFC-LSSVR	94.76%
	K-means	92.38%
	C-means	92.86%
P. A. Kowalski et al.[38]	PNN with LSA	93.4%

2. All the three SVM-related studies used the Gaussian function as the kernel function. Although the Gaussian kernel may be seen as the most commonly used

kernel function in SVMs, other kernel functions, notably polynomial and sigmoid function, can also be used in SVMs and they are worth studying.

3. SVMs involve some problem-dependent hyper-parameters that need to be tuned effectively. What is more, the choice of these hyper-parameters significantly affects the total accuracy of SVMs. In fact, the two LSSVM-related studies [34,37] used one or more optimization algorithms for tuning these hyper-parameters. However, the study reported in [35], which used the standard SVM algorithm, did not involve any optimization process of the SVM hyper-parameters.

An overview of the problem of SVM hyper-parameters optimization, together with a review of the prior work in that area, is presented in the subsequent section.

1.3 SVM Hyper-parameters Optimization

In short, tuning the SVM hyper-parameters is a minimization problem and, generally speaking, two broad choices of the objective function and two broad classes of minimization algorithms exist. The objective function can be either an estimate (like the cross-validation error) of the generalization error or alternatively an upper bound of it. The minimization algorithm can be one of the gradient-based optimization algorithms or one of the stochastic optimization algorithms (also known as meta-heuristic algorithms). Below is the summary of some of the key studies using either type of the objective function and either class of the minimization algorithms.

1.3.1 Related Work

In the literature, many studies have been conducted for tackling this problem. The traditional approach is the grid search [41], which searches only a finite subset of the hyper-parameters' space. This traditional technique chooses a finite number of equally-spaced values for these hyper-parameters, tests all their combinations, and chooses the combination with the least test/validation error. Time consumption is indeed the main drawback of this technique, and it becomes intractable if the number of hyper-parameters are more than two [42].

As an alternative to the traditional method, tuning the SVM hyper-parameters can be considered as a minimization problem, as stated before. This minimization problem searches for the optimum hyper-parameters related to a minimum estimate of the true error (for example, the cross-validation error or holdout testing error) or a minimum upper bound of this error. In general, some studies used deterministic optimization algorithms, such as gradient descent and Newton's method, while others depended on stochastic algorithms, including GA and PSO.

Various analytic upper bounds of the generalization error were proposed in the literature and recommended to be used in the process of tuning the SVM parameters. The most common bounds are the radius/margin bound (it is defined as the ratio of the radius of the sphere enclosing all the data to the separation margin) and the span bound (also known as the span of support vectors) [43]. The main disadvantage of these bounds is that both of them are computationally expensive to compute since they require extra optimization problem to be solved. Another disadvantage is that they are not applicable to all the SVM variants.

In [42,44,45], different versions of the radius/margin bound were used to find the best SVM hyper-parameters. As regards the optimization algorithm, gradient-based algorithms (i.e., the gradient descent and quasi-Newton methods) were used to minimize this bound. These frameworks were applied to multiple benchmark datasets to test its effectiveness.

In another study [46], which is an extension of [42], an estimate of the true error (i.e., the validation error) was used in the optimization problem of the SVM hyper-parameters. The optimization algorithm used in this study was the quasi-Newton optimization algorithm searching for the optimum SVM hyper-parameters that minimized the validation error. As compared to the gradient descent, the quasi-Newton approach shows faster convergence. Moreover, the validation error as the objective function of this minimization problem is, in fact, preferable to the radius/margin bound. The validation error is easy to compute from the validation subset, whereas computing the radius/margin bound requires the solution of another optimization problem.

In addition, multiple class separability criteria; namely kernel-based class separability index [47], the inter-cluster distances in the so-called feature space [48], the distance

between two classes (DBTC) index [49], and the expected square distance ratio (ESDR) [50]; were used in the literature to tune some of the SVM hyper-parameters. The main drawback of these approaches is that they can be used only for tuning the kernel parameters and cannot be used to find the optimum trade-off parameter C.

Using gradient-based optimization algorithms, for instance, gradient descent and Newton's methods, involves two main drawbacks: differentiability and non-convexity. The kernel function and the objective function (either an upper bound or estimate of the true error) must be differentiable and this, unfortunately, is not always guaranteed. In addition to that, gradient-based algorithms suffer mainly from the problem of local minima. In other words, the obtained solution is not guaranteed to be the global solution simply because gradient-based approaches are local optimization techniques. However, tuning the SVM hyper-parameters presents local minima [51] and, therefore, global optimization algorithms are required. Unlike gradient-based methods, stochastic optimization algorithms are global optimization algorithms and they are good alternatives to gradient-based methods.

Accordingly, in the literature, many stochastic optimization algorithms were used for optimizing the SVM hyper-parameters. For example, GA was used widely in the problem of tuning these hyper-parameters [51–55]. Simulated annealing (SA) is another stochastic algorithm which was implemented in the tuning process of the SVM hyper-parameters as well [51,56–58]. PSO was also used in the literature for the purpose of selecting the optimal SVM hyper-parameters [59–62]. Ant colony optimization (ACO) [63,64], ABC [65], TLBO [66,67], CS [68,69], differential evolution (DE) [70,71], and grey wolf optimizer [72] are other famous stochastic optimization algorithms which were used for the same purpose as well.

1.3.2 Limitations of Previous Work

Many previous studies focused on solving the problem of picking the optimal SVM hyper-parameters. These studies used different objective functions and different minimization algorithms, as stated above. However, there are still some limitations to these studies. These limitations can be summarized as follows:

1. Some studies [42–46] used the radius/margin bound or the span bound as the objective function. Nevertheless, these bounds, as mentioned before, are computationally expensive to compute, as they require an additional optimization problem to be solved. On the other hand, the holdout validation error or cross-validation error is an appropriate objective function, since their computation is more straightforward.
2. Some studies [43–46] used one of the gradient-based optimization algorithms to solve the minimization problem that comes up in the tuning process of the SVM hyper-parameters. Unfortunately, these methods are local optimization approaches and the achieved solution may be a local minimum and not a global minimum. Furthermore, they require both the objective function and kernel function to be differentiable; sometimes, this is not easy to achieve and an additional smoothing process of the objective function is necessary. As a result, meta-heuristic optimization algorithms are preferable because they are global optimization techniques and they do require neither the kernel function nor the objective function to be differentiable.
3. A number of the recently proposed meta-heuristic algorithms have not yet been applied to the SVM model selection.
4. Almost all the aforementioned studies optimized only the Gaussian kernel. However, SVMs are a rich algorithm, which offers multiple choices of the kernel functions. For instance, sigmoid and polynomial kernels are other famous kernel functions. Also, the hybrid of two kernel functions is another choice. To the best of our knowledge, SVMs with the sigmoid kernel have not been optimized in the literature using stochastic optimization algorithms. One point worth noting is that the computational burden of sigmoid SVMs is comparable to that of the Gaussian kernel and less than that of the polynomial kernel.

CHAPTER 2
SUPPORT VECTOR MACHINES

The previous chapter presents a brief introduction into the problem of optimizing the SVM hyper-parameters. In addition, a literature review about the wheat varieties dataset, which is, in fact, the benchmark dataset of this thesis, was conducted. This chapter presents a thorough explanation of SVMs, as a learning algorithm, together with the mathematical foundation of this powerful algorithm.

Classical learning algorithms, such as neural networks, essentially depend upon the empirical risk minimization (ERM) principle. In contrast, the SVM algorithm is based upon the structural risk minimization (SRM) principle. The ERM principle, in essence, minimizes the empirical error, or rather the training error, so as to have the learning algorithm learn the dataset at hand. The SRM principle, on the other hand, attempts to minimize an upper bound of the test error instead of minimizing the so-called empirical error. This bound is the sum of the empirical error and confidence interval. An SRM-based algorithm, like SVMs, minimizes this bound by finding the best trade-off or balance between the training error and confidence interval. Indeed, there is an essential trade-off between the classifier complexity and the training error. For a given amount of training data, whenever the classifier complexity decreases, the confidence interval decreases but the training error increases as well. Hence, the right balance between the value of the training error and the amount of the classifier complexity (or confidence interval) ought to be achieved, and that is the main goal of the SRM principle. In fact, minimizing the empirical error ensures good generalization ability (low probability of the test error) as long as the amount of the available training data is large enough; therefore, the ERM principle is suitable when large training data are available. On the other hand, minimizing this upper bound, which can be considered as the sum of the confidence interval and empirical error, ensures a good generalization ability even though the training data are

small. Therefore, SVMs possesses high generalization ability due to their SRM principle. More details about the SRM principle and its mathematical analysis can be found in [73].

In their original form, SVMs, as a pattern recognition algorithm, is a binary (i.e., two-class) classifier of linearly separable data. Linearly separable data refer to the fact that the data can be separated with no error using a linear separating surface (i.e., a straight line in the case of the two-dimensional space, a flat plane in the case of the three-dimensional space and hyperplane in higher-dimensional spaces). Separating the training data without error means that the training error is zero. Yet, here arise the following questions:

1. What if the data are noisy and non-separable (i.e., the data cannot be separated with no error)?
2. What if the dataset at hand happens to be not linearly separable (i.e., a linear separating surface cannot separate the data apart)?
3. What if the classification problem is not binary and there exist more than two classes (i.e., multi-class classification)?

Section 2.1 explains the SVM algorithm in its original form that is used for two-class linearly separable data. Then, Section 2.2 answers the first question and extends the SVM algorithm to be used with even non-separable noisy data. Next, Section 2.3 addresses the issue raised in the second question and extends the SVM algorithm more to be used even with nonlinearly separable data. Section 2.4 summarizes the generalized SVM algorithm that can be universally used, regardless of whether the dataset in question is separable and regardless of whether that dataset is linearly or nonlinearly separable. Finally, Section 2.5 describes how SVMs can be extended into multi-class classification problems.

2.1 Original SVM Algorithm

The main idea of SVMs in their original version is to separate two-class data apart (the first class is termed as the *positive class* while the second class is termed as the *negative class*) [73,74]. They primarily search for the optimal decision hypersurface[1] (it is, in fact, the optimal separating hyperplane in the special case of linearly separable data) that can

[1] Hypersurface is a curve in the case of the two-dimensional space, surface in the case of the three-dimensional space and hypersurface in higher-dimensional spaces.

separate the negative class from the positive class with the maximal possible margin between the two classes. Figure 2.1 shows that optimal separating hyperplane in the case of a two-dimensional space (i.e., plane) where the data consist of only two features. In this case, the decision surface is a line. The negative-class instances in the figure are drawn as rectangles while the positive-class instances are drawn as circles. The support vectors (the data on the boundary) are colored in grey.

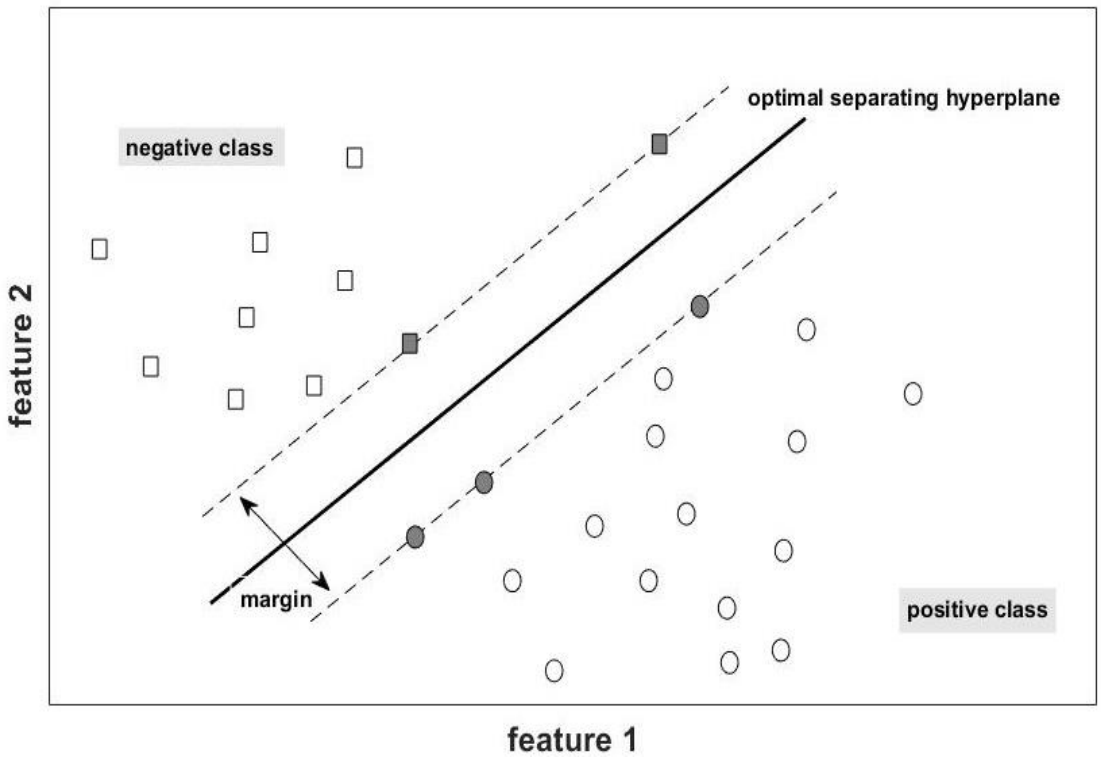

Figure 2.1 Optimal separating hyperplane (circles represent the positive class, rectangles represent the negative class, and grey circles and rectangles represent the support vectors)

2.1.1 Mathematical Formation of the Algorithm

Mathematically speaking, let us have the following training pairs:

$$(\boldsymbol{x}_1, y_1), \cdots, (\boldsymbol{x}_n, y_n)$$

where $\boldsymbol{x}_i \in R^d$, $y_i \in \{-1,1\}$ and i = 1,…,n.

Thus, n is considered as the training data size. The input space is d-dimensional which means the input data contain d features. The data point $\mathbf{x}_i$ belongs to the negative class if y_i = -1; otherwise, it belongs to the positive class since y_i = 1.

According to SVMs, these data can be separated by the hyperplane defined by:

$$\boldsymbol{w} \cdot \boldsymbol{x} + b = 0 \tag{2.1}$$

So SVMs search for the parameter vector **w** in addition to the threshold b (sometimes called bias) of the optimal hyperplane provided that

$$\boldsymbol{w} \cdot \boldsymbol{x_i} + b \geq 1 \quad when\ y_i = 1 \tag{2.2}$$

and

$$\boldsymbol{w} \cdot \boldsymbol{x_i} + b \leq -1 \quad when\ y_i = -1 \tag{2.3}$$

The last two inequalities can be written both at once as:

$$y_i(\boldsymbol{w} \cdot \boldsymbol{x_i} + b) \geq 1 \tag{2.4}$$

The hyperplane is in its canonical form once it satisfies Inequality (2.4). This inequality also ensures that the training error is zero, and that literally means all the training data are separated with no error.

The data points that satisfy the following equality:

$$y_i(\boldsymbol{w} \cdot \boldsymbol{x_i} + b) = 1 \tag{2.5}$$

are coined as *support vectors*. Specifically speaking, the data points on the boundary of the margin are considered support vectors and these so-called support vectors are normally a small proportion of the training data [73].

Once the algorithm has found the optimal hyperplane, the decision function or rule is

$$f(x) = sign(\boldsymbol{w} \cdot \boldsymbol{x} + b) \tag{2.6}$$

2.1.2 Constructing the Optimal Hyperplane: an optimization problem

The hyperplane separating the data with the largest margin is deemed the optimal hyperplane. Therefore, we need to maximize the margin. The margin ρ of the hyperplane defined by Eq. (2.1) is:

$$\rho = \frac{2}{\|\boldsymbol{w}\|^2} \tag{2.7}$$

Accordingly, maximizing the margin is equivalently achieved by minimizing the norm of the parameter vector **w**. In other words, the search for the optimal hyperplane becomes the following optimization problem:

$$\min_{\boldsymbol{w}} \frac{1}{2}\boldsymbol{w} \cdot \boldsymbol{w} \tag{2.8}$$

subject to Inequality (2.4).

In fact, $\boldsymbol{w} \cdot \boldsymbol{w}$ is just the dot product of **w** with itself which is equal to the square of the norm of this vector $\|\boldsymbol{w}\|^2$.

Equation (2.8) is a constrained quadratic programming problem subject to Inequality (2.4). So, the above equation is just a convex optimization problem. What is more, this particular problem can be solved using quadratic programming.

2.1.3 Simpler Form of the Optimization Problem

The aforementioned constrained quadratic programming problem can be made simpler. Using Lagrange multipliers, the constrained optimization problem ought to be transformed first into an unconstrained optimization problem in order to define the following Lagrange functional:

$$L(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{w}.\boldsymbol{w} - \sum_{i=1}^{n} \alpha_i [y_i(\boldsymbol{w} \cdot \boldsymbol{x_i} + b) - 1] \tag{2.9}$$

where α_i are nonnegative and this vector of values is known as Lagrange multipliers.

Then, the saddle point is the optimal solution of the Lagrange functional $L(\boldsymbol{w}, b, \boldsymbol{\alpha})$. In other words, the Lagrange functional must be maximized with respect to $\boldsymbol{\alpha}$ and minimized with respect to b and **w** in order to figure out the optimum solution. Then, the optimum solutions $\mathbf{w_0}$, b_0 and $\boldsymbol{\alpha^0}$, as the saddle point, satisfy the following conditions:

$$\frac{\partial L(\boldsymbol{w_0}, b_0, \boldsymbol{\alpha^0})}{\partial \boldsymbol{w}} = 0 \tag{2.10}$$

$$\frac{\partial L(\boldsymbol{w_0}, b_0, \boldsymbol{\alpha^0})}{\partial \boldsymbol{b}} = 0 \tag{2.11}$$

From the first condition (i.e., Eq. (2.10)), the following equality can be deduced:

$$\sum_{i=1}^{n} \alpha_i \, y_i = 0 \tag{2.12}$$

In addition, the following equality can be deduced from the second condition (i.e., Eq. (2.11))

$$\boldsymbol{w} = \sum_{i=1}^{n} \alpha_i \, y_i \boldsymbol{x}_i \tag{2.13}$$

Thus, the parameter vector of the optimal hyperplane $\boldsymbol{w}$ is just a linear combination of $\mathbf{x}_i$ (in other words, the training input vectors) weighted by the corresponding α_i (i.e., the Lagrange multipliers)

The Kuhn-Tucker conditions state that the product of the constraints and their corresponding optimum Lagrange multipliers is equal to zero:

$$\alpha_i^0 [y_i(\boldsymbol{w}_0 \cdot \boldsymbol{x}_i + b_0) - 1] = 0 \quad for\ i = 1, \cdots, n \tag{2.14}$$

From this last condition, it can be concluded that only the support vectors, which satisfy Eq. (2.5), have nonzero Lagrange multipliers α_i while all the remaining training data have zero Lagrange multipliers. If this conclusion is applied to Eq. (2.13), only the support vectors participate in constructing the optimal hyperplane and finding the parameter vector $\boldsymbol{w}$.

Substituting Eq. (2.12) and Eq. (2.13) into the Lagrange functional $L(\boldsymbol{w}, b, \boldsymbol{\alpha})$, one obtains the following functional:

$$W(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \boldsymbol{x_i} \cdot \boldsymbol{x_j} \tag{2.15}$$

Then, the optimization problem becomes

$$\max_{\boldsymbol{\alpha}} W(\boldsymbol{\alpha}) \tag{2.16}$$

Subject to these two constraints:

$$\sum_{i=1}^{n} \alpha_i \, y_i = 0 \tag{2.17}$$

$$and \qquad \alpha_i \geq 0 \quad for\ i = 1, \cdots, n \tag{2.18}$$

The optimization problem with this simplified version is still considered as a quadratic optimization problem. Yet, it is simpler because of its simpler constraints.

2.2 Soft-margin SVMs: Noisy Data

The previous section described the original SVMs that can be used with separable data. Now, what if the data are non-separable? In other words, if the data are noisy and they cannot be separated without error, can SVMs still address this issue? This section addresses this case and introduces a generalization of SVMs into non-separable data.

Normally, the separable case of SVMs is termed as *hard-margin SVMs*, whereas the non-separable case is termed as *soft-margin SVMs*. Concerning the hard-margin case, all training data points lie on their correct side beyond the margin or at least on the boundary and no any data point is allowed to reside on the opposite side of the separating hyperplane or even inside the margin. On the other hand, some data can reside on the opposite side or even on the margin in the soft-margin case as shown in Figure 2.2.

In Figure 2.2, ξ_i is a slack variable ($\xi_i \geq 0$ for i=1,…,n). This slack variable, which is a distance measure, can be used as an indicator of the training error for each data point.

The extension of original SVMs to accommodate the non-separable case can be done by modifying two things in the original SVMs:

1. The objective function in Eq. (2.8) need to be changed into the following equation:

$$\min_{\boldsymbol{w},\xi} \frac{1}{2}\boldsymbol{w}\cdot\boldsymbol{w} + C\sum_{i=1}^{n}\xi_i^{\sigma} \tag{2.19}$$

where C is known here as the *trade-off parameter* or interchangeably the *penalty constant*. In fact, C acts as a regularization parameter such that it controls the trade-off between the complexity of the algorithm and the training error. This parameter, therefore, has a major role in determining the overall SVM performance since a good value of C ensures that the data are neither underfitted nor overfitted. Also, σ is the exponent of the slack variable ξ_i. Usually, the most commonly used values of σ are 1 or 2. When σ is equal to 1, it is called *L1 soft-margin SVMs.* Similarly, it is called *L2*

soft-margin SVMs when σ is equal to 2. In the L1 soft-margin SVMs (σ = 1), the involved computation is simple as compared to the other values of σ [73]. Therefore, the L1 soft-margin SVMs will be considered in the rest of this section.

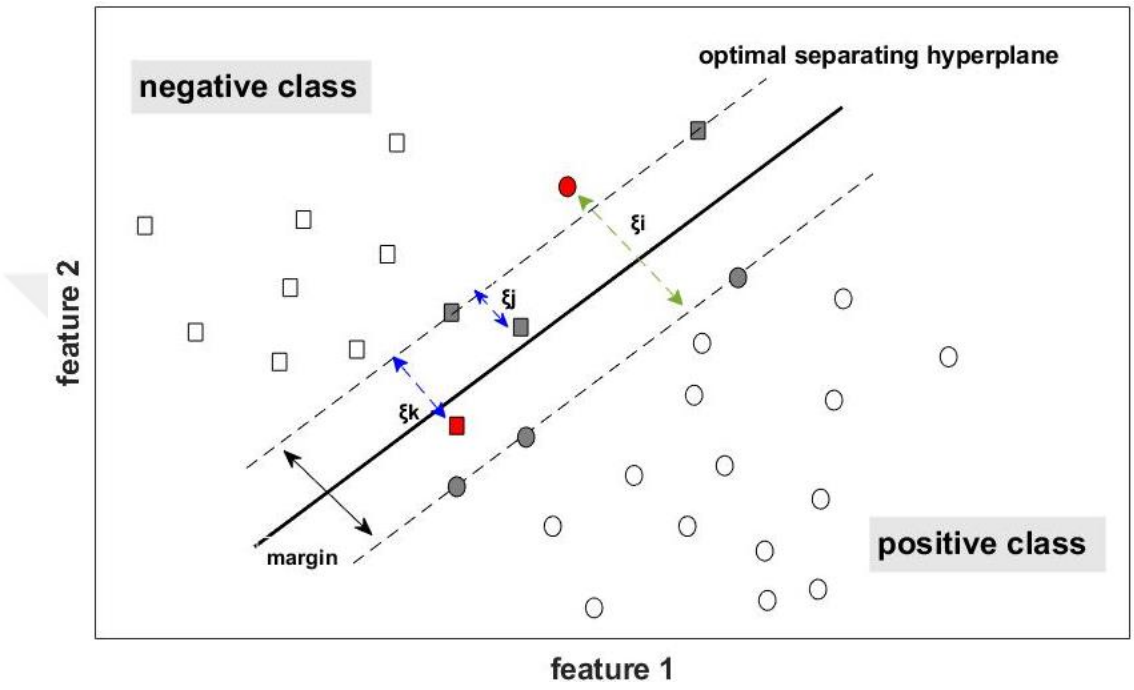

Figure 2.2 Optimal separating hyperplane in the soft-margin case (the support vectors are colored in grey while the misclassified support vectors are colored in red).

2. The inequality (2.4) is changed into the following inequality:

$$y_i(\boldsymbol{w} \cdot \boldsymbol{x_i} + b) \geq 1 - \xi_i, \quad i = 1, \ldots, n \tag{2.20}$$

According to this inequality, if $0 < \xi_i < 1$, then $\mathbf{x_i}$ happens to be correctly classified but not with a maximal margin (i.e., $\mathbf{x_i}$ is inside the margin but on its right side of the separating hyperplane). Furthermore, $\mathbf{x_i}$ is misclassified if $\xi_i \geq 1$. Therefore, the slack variable ξ_i can be indeed considered as an indicator of the training error, which means that minimizing ξ_i actually minimizes the training error.

In short, the minimization problem in the soft-margin SVMs is:

$$\min_{\boldsymbol{w},\xi} \frac{1}{2}\boldsymbol{w}\cdot\boldsymbol{w} + C\sum_{i=1}^{n}\xi_i \tag{2.21}$$

Subject to the subsequent inequality:

$$y_i(\boldsymbol{w}\cdot\boldsymbol{x_i} + b) \geq 1 - \xi_i, \quad i = 1, \dots, n \tag{2.22}$$

where $\xi_i \geq 0$ for i = 1,…,n.

In fact, if one follows the same steps taken in the hard-margin case in the previous section, they will derive almost the same equations shown in the previous section. Similar to the hard-margin case, a simple optimization problem can be derived as in the hard-margin case and then the following functional must be maximized with regard to $\boldsymbol{\alpha}$:

$$W(\boldsymbol{\alpha}) = \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{n}\alpha_i\alpha_j y_i y_j \boldsymbol{x_i}\cdot\boldsymbol{x_j} \tag{2.23}$$

Subject to the following constraints:

$$\sum_{i=1}^{n}\alpha_i\, y_i = 0 \tag{2.24}$$

$$and \qquad 0 \leq \alpha_i \leq C \qquad for\ i = 1, \cdots, n \tag{2.25}$$

The constraints of the optimization problems are the only difference between the soft-margin SVMs and hard-margin SVMs. The Lagrange multipliers $\boldsymbol{\alpha}$ are upper bounded by the trade-off parameters C in the soft-margin SVMs. One particular point worth noticing is that the hard-margin SVMs can be considered as just a special case of the soft-margin SVMs where C is equal to infinity. The following equation:

$$\boldsymbol{w} = \sum_{i=1}^{n}\alpha_i\, y_i \boldsymbol{x}_i \tag{2.26}$$

holds in the soft-margin SVMs as well.

2.3 Nonlinear Decision Surfaces: Nonlinearly Separable Data

Until now, we understand how to deal with the non-separability problem as long as the optimal separating hypersurface is linear (i.e., a hyperplane). However, what if the optimal separating hypersurface is not linear (the data are not linearly separable), and,

instead, they can be separated efficiently using a nonlinear hypersurface? In fact, the genuine separating hypersurfaces are not always hyperplanes but they are often nonlinear hypersurfaces [75]. This section sheds light on this nonlinearity problem where a nonlinear hypersurface is preferable to a hyperplane.

The main idea of solving this problem is:

Whenever the dataset is separable in the input space using a nonlinear hypersurface, there should be a higher-dimensional space known as the feature space in which the images of data can be separated using a hyperplane; that is, the dataset is linearly separable in the feature space.

As a result, the data have to be transformed from their original input space R^d into another remarkably higher-dimensional feature space R^f by means of nonlinear mapping. Let $\Phi(\mathbf{x})$ be this nonlinear mapping and defined as:

$$\Phi(\boldsymbol{x}) = (\varphi_1(\boldsymbol{x}), \varphi_2(\boldsymbol{x}), \cdots, \varphi_f(\boldsymbol{x}))$$

such that $\Phi(\boldsymbol{x})$ is defined as the image of $\mathbf{x}$ in the so-called feature space.

Then, the hyperplane in the feature space is defined as:

$$\boldsymbol{w} \cdot \Phi(\boldsymbol{x}) + b = 0 \tag{2.27}$$

provided that, for i = 1,…,n,

$$y_i(\boldsymbol{w} \cdot \Phi(\boldsymbol{x_i}) + b) \geq 1 \tag{2.28}$$

Then the decision function becomes:

$$f(x) = sign(\boldsymbol{w} \cdot \Phi(\boldsymbol{x}) + b) \tag{2.29}$$

Now, the above equations form the SVM algorithm for nonlinearly separable data. In essence, the problem is still the same except that the training data are transformed into the feature space $\Phi(\boldsymbol{x}_i)$, and then their images are used instead of the data points themselves $\mathbf{x_i}$. Consequently, the images of the data have to be found first. Nevertheless, the mapping function $\Phi(\mathbf{x})$ is not easy to find because the dimensionality of feature space is significantly high. Having said that, there exists some clever trick to tackle this problem. This trick is called the *kernel trick* and the subsequent subsection elaborates upon that.

2.3.1 The Kernel Trick

As stated before, the algorithm formation of the nonlinear case is the same as the linear case (the original SVMs described in Section 2.1) except that the images of the data are used instead of the data themselves. Therefore, if one follows the same linear-SVM-related procedures, they will definitely arrive at the same equations. So, in the feature space, the parameter vector **w** becomes:

$$\boldsymbol{w} = \sum_{i=1}^{n} \alpha_i \, y_i \Phi(\boldsymbol{x}_i) \tag{2.30}$$

If Eq. (2.30) is put into Eq. (2.29), the decision function will be:

$$f(\boldsymbol{x}) = sign(\sum_{i=1}^{n} \alpha_i \, y_i \Phi(\boldsymbol{x}_i) \cdot \Phi(\boldsymbol{x}) + b) \tag{2.31}$$

Also, the simple functional, that is needed to be maximized with respect to Lagrange multipliers, will be

$$W(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \Phi(\boldsymbol{x_i}) \cdot \Phi(\boldsymbol{x_j}) \tag{2.32}$$

Subject to

$$\sum_{i=1}^{n} \alpha_i \, y_i = 0 \tag{2.33}$$

$$and \qquad 0 \leq \alpha_i \leq C \qquad for\ i = 1, \cdots, n \tag{2.34}$$

such that C is taken as infinity in the hard-margin case.

As shown in the two most important equations of the SVM algorithm (i.e., Eq. (2.31) and Eq. (2.32)), they always involve the dot product of the images of the observations or data points. Accordingly, the main idea of the kernel trick is that only the dot product of the images of the data points are required, and the explicit expression of the mapping function $\Phi(\boldsymbol{x}_i)$ is not necessary. Moreover, computing the dot product of these images in the feature space is considered as an exact equivalent to the computation of the kernel function of the data themselves in the input space. As a mathematical form, the following equality:

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \Phi(\boldsymbol{x}_i) \cdot \Phi(\boldsymbol{x}_j) \tag{2.35}$$

holds such that the quantity $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is the *kernel function* of the two data vectors $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$.

Therefore, Eq. Eq. (2.31) and Eq. (2.32) can be equivalently expressed as:

$$f(\boldsymbol{x}) = sign(\sum_{i=1}^{n} \alpha_i \, y_i K(\boldsymbol{x}_i, \boldsymbol{x}_j) + b) \tag{2.36}$$

and

$$W(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) \tag{2.37}$$

Hence, the only difference between the linear and nonlinear SVMs is that we have to define an appropriate kernel function in the nonlinear case. In fact, the nonlinear case is the general case such that it comes back to the linear case if the linear kernel function is chosen. Different nonlinear decision hypersurfaces in the input space can be constructed by different kernel functions. Many different kernel functions exist in the literature; however, the choice of the kernel function significantly affects the performance of SVMs. The most common kernel functions are shown in Table 2.1.

These popular kernel functions are:

1. **Linear kernel:** linear SVMs use this linear kernel, which is only the dot product of the data points. This kernel does not contain any parameter and the feature space is indeed the input space itself.
2. **Polynomial kernel:** this kernel is a rich kernel as the polynomial degree r can be changed to generate different variants of the polynomial kernel. For instance, when r is equal to 1, it becomes almost the same as the linear kernel. Also, when r is equal to 2, the polynomial kernel is called a quadratic kernel and the separating hypersurface is a quadratic hypersurface (e.g., a parabola in two-dimensional space). Likewise, when r is equal to 3, the polynomial kernel is called a cubic kernel and so on. The only kernel parameter in the polynomial kernel is its degree r, which should be a positive integer.
3. **Gaussian kernel:** this kernel function is also known in the literature as the radial basis function (RBF). It is, in fact, considered as the reasonable first choice [76].

The Gaussian kernel contains one kernel parameter σ known as the *kernel width* or *scale* ($\sigma > 0$).

4. **Sigmoid kernel:** in this kernel function, there exist two kernel parameters. These parameters are the kernel scale γ, which should be a positive number, and the threshold or offset parameter δ, which should be a small positive number [77]. In fact, there is an equivalence between a two-layer neural network and this kernel such that:
   - **i.** the number of support vectors is equivalent to the number of the hidden neurons.
   - **ii.** the weights of the first layer are just the support vectors themselves.
   - **iii.** the Lagrange multipliers $\boldsymbol{\alpha}$ represent the weights of the second layer**.**

Table 2.1 Popular kernel functions

Kernel Type	Mathematical Form	Kernel Parameters
Linear	$K(\boldsymbol{x_i}, \boldsymbol{x_j}) = \boldsymbol{x_i} \cdot \boldsymbol{x_j}$	-
Polynomial	$K(\boldsymbol{x_i}, \boldsymbol{x_j}) = (\boldsymbol{x_i} \cdot \boldsymbol{x_j} + 1)^r$	r: polynomial degree
Gaussian	$K(\boldsymbol{x_i}, \boldsymbol{x_j}) = exp(-\frac{1}{2\sigma}\|\boldsymbol{x_i} - \boldsymbol{x_j}\|^2)$	σ: kernel width or scale
Sigmoid	$K(\boldsymbol{x_i}, \boldsymbol{x_j}) = \tanh(\gamma \boldsymbol{x_i} \cdot \boldsymbol{x_j} - \delta)$	γ: kernel scale δ: threshold parameter

One important point about the kernel function is its parameters. The trade-off parameter C, together with the kernel parameters, are also called *SVM hyper-parameters* since they should be chosen properly before training. The values of these hyper-parameters significantly influence the SVM performance. Nonetheless, they mainly rely upon the problem at hand where they have to be tuned efficiently according to the dataset through an optimization process.

2.4 The Generalized SVM Algorithm

The generalized SVM algorithm can be summarized as:

1. the appropriate kernel function is selected
2. the hyper-parameters (the kernel parameters and trade-off parameter) are selected appropriately according to the given dataset.
3. the following constrained quadratic programming problem must be worked out:

$$\max_{\boldsymbol{\alpha}} W(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) \tag{2.38}$$

Subject to:

$$\sum_{i=1}^{n} \alpha_i \, y_i = 0 \tag{2.39}$$

$$and \qquad 0 \leq \alpha_i \leq C \qquad for\ i = 1, \cdots, n \tag{2.40}$$

4. After the optimal Lagrange multipliers $\boldsymbol{\alpha}$ are found, then the decision function is defined according to:

$$f(\boldsymbol{x}) = sign(\sum_{i=1}^{n} \alpha_i \, y_i K(\boldsymbol{x}_i, \boldsymbol{x}) + b) \tag{2.41}$$

such that, in the feature space, the optimal hyperplane is defined as:

$$\sum_{i=1}^{n} \alpha_i \, y_i K(\boldsymbol{x}_i, \boldsymbol{x}) + b = 0 \tag{2.42}$$

Important Notes:

(Note I): the hard-margin SVM can be achieved by making C equal to infinity.

(Note II): the linear SVM can also be achieved by choosing the linear kernel function.

(Note III): One popular method for solving this SVM quadratic programming problem is the sequential minimal optimization (SMO) [78].

2.5 Multi-class SVMs

So far, SVMs are used only as binary (two-class) classifier. However, many real problems are multiclass classification problems and, therefore, SVMs need to be extended to include the multiclass case. This section shows how that can be done. In fact, SVMs can

be generalized to be used as a multiclass classification algorithm via two main approaches [1]:

1. One-versus-all technique
2. One-versus-one technique

In addition, there is still a third method which is a general case of the two above-mentioned methods. This method is known as Error-correcting output code (ECOC). For brevity purposes, just the two main methods will be explained below (for curious readers, more details about ECOC can be found in [79]).

2.5.1 One-versus-all Method

In this approach, the m-class classification problem (i.e., there exist m classes) is transformed into m binary classification problems. Thus, m binary (two-class) SVM classifiers are generated in this method. The first classification problem is to separate the first class from the remaining classes such that the first class represents the positive class whilst all the data belonging to the remaining classes construct the negative class. Likewise, the second class represents the positive class of the second classification problem and the negative class is all the remaining classes combined and so on. Since m binary SVM classifiers exist, there are m corresponding score functions. The *score function* of the jth classifier is :

$$S_j(\boldsymbol{x}) = \sum_{i=1}^{n} \alpha_i^j \, y_i K(\boldsymbol{x}_i, \boldsymbol{x}) + b^j \tag{2.43}$$

such that the quantity $K(\boldsymbol{x}_i, \boldsymbol{x})$ is the kernel function (for linear SVM, it is the linear kernel) and $\boldsymbol{\alpha^j}$ and b^j are the parameters of the jth binary SVM classifier.

In fact, the decision function f(x), as defined in Eq. (2.41), is the sign of the score function S($\boldsymbol{x}$). The data point $\boldsymbol{x}$ will be then classified into the positive class whose corresponding classifier has the maximum score function and, in this multiclass case, the decision function becomes:

$$f(\boldsymbol{x}) = \underset{j=1,\ldots,m}{\operatorname{argmax}} \; S_j(\boldsymbol{x}) \tag{2.44}$$

2.5.2 One-versus-one Method

The one-versus-one method is interchangeably called the pairwise method in the literature. It converts an m-class classification problem into $\frac{m(m-1)}{2}$ binary classification problems such that each two-class classification problem involves only two classes of the available classes. Consequently, it is called a pairwise method. The one-versus-one approach, therefore, generates $\frac{m(m-1)}{2}$ binary (two-class) SVM classifiers where, for example, the first binary SVM classifier takes into account only the first and second classes ignoring all the remaining classes.

Mathematically speaking, there exist $\frac{m(m-1)}{2}$ binary SVM classifiers where their corresponding decision functions are defined as:

$$f_{ij}(\boldsymbol{x}) = sign\left(\sum_{k=1}^{n} \alpha_k^{ij}\, y_k K(\boldsymbol{x}_k, \boldsymbol{x}) + b^{ij}\right) for\ i,j = 1, \dots, m;\ i \neq j \qquad (2.45)$$

where $\boldsymbol{\alpha^{ij}}$ and b^{ij} are the parameters of the binary SVM classifier ij and this classifier separates the ith class from the jth class. Also, $f_{ij}(\boldsymbol{x})$ is the same as $f_{ji}(\boldsymbol{x})$, so only one of them is considered and found.

In the one-versus-one method, the classification of any data point $\boldsymbol{x}$ is done through a voting process such that every class will have a score or vote. The vote of the ith class is:

$$V_i(\boldsymbol{x}) = \sum_{j=1, j\neq i}^{m} H[\, f_{ij}(\boldsymbol{x})\,] \qquad (2.46)$$

where H[·] is the Heaviside step function. The data point $\boldsymbol{x}$ is then assigned into the class whose vote is the highest, and the final decision function is defined as:

$$f(\boldsymbol{x}) = \underset{i=1,\dots,m}{\text{argmax}}\ V_i(\boldsymbol{x}) \qquad (2.47)$$

2.5.3 Comparison Between the Two Methods

In the one-versus-all method, all the training data are used with every binary SVM classifier and the number of the binary SVM classifiers equals the number of classes (i.e., equal to m). On the other hand, the one-versus-one method uses only the data of two classes as training data for each binary SVM classifier. Consequently, in the one-versus-

one approach, the amount of training data is smaller than the amount of training data in the one-versus-all approach. That, indeed, is an advantage which makes the one-versus-one method is preferable to the one-versus-all method. However, the number of binary SVM classifiers, in the one-versus-one method, is $\frac{m(m-1)}{2}$ instead of m. Thus, the one-versus-one method is faster than the one-versus-all method, particularly when there is a small number of available classes.

In this work, multiclass SVMs are applied to the wheat varieties dataset, which has only three classes. In the three-class case, the number of the binary SVM classifiers is the same in both methods so the one-versus-one method is definitely preferred.

CHAPTER 3

META-HEURISTIC OPTIMIZATION ALGORITHMS

As stated in the first chapter, SVMs contain hyper-parameters substantially affecting the overall performance of SVMs. Moreover, these hyper-parameters rely mainly on the dataset at hand. Therefore, in order to improve the performance, the optimal values of these hyper-parameters should be chosen beforehand using an optimization scheme. In general, two broad optimization methods exist. The first type is grouped as the *deterministic optimization approaches,* while the second type is grouped as the *meta-heuristic optimization approaches*.

3.1. Overview of Meta-heuristic Optimization Algorithms

Meta-heuristic optimization algorithms are also known as *stochastic optimization algorithms* due to the fact that, unlike their deterministic counterparts, these algorithms involve some random search of the optimal solution. Like deterministic algorithms, these algorithms are iterative algorithms consisting of many iterations. Moreover, meta-heuristic algorithms are nature-inspired algorithms where different meta-heuristic algorithms are inspired by different natural behaviors related to natural creatures, laws, or phenomena.

Too many meta-heuristic optimization algorithms exist in the literature. Some of the most popular ones are Genetic Algorithm (GA)[80], Particle Swarm Optimization (PSO)[81], Ant Colony Optimization (ACO)[82], and Artificial Bee Colony (ABC)[83]. Although many meta-heuristic algorithms have been developed in the literature, there is no one algorithm that is the best for all optimization problems. This is what is known as the *No Free Lunch theorem* [84]. In other words, one algorithm can show excellent results in some optimization problems while the same algorithm shows bad results in other problems.

Two main types of meta-heuristic optimization algorithms exist, and they are:

1. **Evolutionary algorithms:** meta-heuristic algorithms that are inspired by natural evolution. One popular example of this type is GA.
2. **Swarm-intelligence-based algorithms:** meta-heuristic algorithms which are inspired by the intelligent behavior of swarms, flocks, or herds. Two important examples of this type are PSO and ABC.

In meta-heuristic optimization algorithms, two important concepts arise. These concepts are *exploration* (sometimes called *diversification*) and *exploitation* (sometimes called *intensification*). Exploration refers to the capability of the algorithm to globally search for promising regions in the search space. Exploitation, however, refers to the capability of the algorithm to locally search these promising regions found in the exploration process in order to find local optima inside these regions. There is normally a trade-off between exploration and exploitation, and a good meta-heuristic algorithm is the one that maintains a good trade-off between exploration and exploitation.

Meta-heuristic optimization algorithms contain control parameters, and their values significantly affect the convergence and performance of these algorithms. In general, these control parameters divide into two classes: *general control parameters* and *algorithm-specific control parameters*. All meta-heuristic algorithms share the same general control parameters, namely the population size and the termination criterion (the termination criterion can be the maximum number of iterations, maximum number of objective function evaluations or tolerance). Algorithm-specific control parameters vary from one algorithm to another. In fact, each algorithm has its own algorithm-specific control parameters, for instance the parameter *limit* in ABC. The fewer the algorithm-specific control parameters are, the better the algorithm is since only a few parameters need to be tuned.

3.1.1 Comparison Between Deterministic and Meta-heuristic Algorithms

The deterministic optimization approaches, also known as the traditional optimization approaches, are mainly gradient-based, for example gradient descent and Newton's methods. Thus, the derivatives of the objective function have to be known. Nonetheless, the derivatives of the objective function are sometimes unknown or difficult to compute.

Further, these traditional optimization methods are known as *local optimization techniques* since the solution found by one of these techniques may be a local optimum and not the global optimum. So, these methods are not suitable for problems with multiple local optima. In contrast, the meta-heuristic optimization methods are *global optimization techniques* and they can be applied to real problems with multiple local optima. In addition, the derivatives of the objective function are not required in these methods and they are simple to implement. To sum up, the advantages of the meta-heuristic optimization algorithms over the deterministic optimization algorithms are:

1. They are simple and easy to implement and learn
2. They are derivative-free and no need for the derivatives of the objective function. Moreover, unlike the traditional optimization methods, the mathematical formula of the objective function is not required either.
3. They can deal with even discontinuous objective functions.
4. They are global optimization algorithms

For those reasons, the optimization of the SVM hyper-parameters is done in this work using meta-heuristic optimization algorithms. According to the No Free Lunch theorem, there is no one meta-heuristic optimization algorithm that always yields good results whatever the optimization problem is. Thus, five meta-heuristic optimization algorithms are used in this work to tune the SVM hyper-parameters according to the wheat varieties dataset and their results are compared. These algorithms are ABC, ABC with depth-first-search frameworks (DFSABC), Teaching-learning-based Optimization (TLBO), Teaching-learning-based ABC (TLABC), and Cuckoo Search (CS). All these algorithms are population-based algorithms, which means, in each iteration, a population of solutions are generated (this group of solutions is named a generation). The subsequent section describes the general standard formation of an optimization problem while the remaining sections explain these five algorithms.

3.2 Optimization Problem Formation

The standard optimization problem is defined as a minimization problem as follows:

$$\min_{\boldsymbol{x}} \ f(\boldsymbol{x}) \tag{3.1}$$

where $f(\boldsymbol{x})$ is called the *objective function* (also called the cost function in the minimization case), and $\boldsymbol{x}$ is a vector containing one or more variables, which are called the *design variables* or *optimization parameters*.

If $f(\boldsymbol{x})$ should be maximized rather than minimized, then a minus sign is added in front of $f(\boldsymbol{x})$ in Eq. (3.1). That is, minimizing $-f(\boldsymbol{x})$ is equivalent to maximizing $f(\boldsymbol{x})$.

3.3 Artificial Bee Colony (ABC)

ABC is one of the most popular swarm-intelligence-based optimization algorithms [83]. It is a nature-inspired algorithm that mimics the intelligent behavior of honeybees. In particular, it imitates the foraging process of honeybees.

In ABC, the agents searching the problem space are called *artificial bees*. In fact, there are three kinds of these artificial bees: *employed bees*, *onlooker bees,* and lastly, *scout bees*. Accordingly, the total population, also known as the *colony*, is divide into employed bees (represent 50% of the colony or population), onlooker bees (represent 50% of the colony) and one scout bee (which is one of the original employed bees). The food sources that artificial bees search for represent potential solutions to the optimization problem. In addition, the nectar amount of any food source represents the fitness of the solution, which is inversely proportional to the value of the objective function at this solution.

The ABC algorithm consists of four phases: the *initialization phase*, the *employed bee phase*, the *onlooker bee phase,* and finally, the *scout bee phase*.

3.3.1 initialization Phase

This phase happens only once at the beginning of the algorithm. In this phase, SN initial random food sources (i.e., food sources represent solutions) are generated according to:

$$X_{i,j}^{0} = L_j + r_j \left(U_j - L_j\right) \quad for\ i = 1, \dots, SN\ and\ j = 1, \dots, D \tag{3.2}$$

where SN is the number of food sources, D is the dimensionality of the search space (the number of optimization parameters or variables), $X_{i,j}^{0}$ is the jth dimension of the ith initial food source, L_j and U_j are the lower and upper bounds of the jth dimension, and r_j is a

random number (i.e., $r_j \in [0,1]$). The number of food sources SN is equal to the number of employed bees (also equal to the number of onlooker bees), which is equal to half of the population size N. Then, the fitness of the i[th] food source X_i is calculated as follows:

$$fitness(X_i) = \begin{cases} \dfrac{1}{1+f(X_i)} & if \;\; f(X_i) \geq 0 \\ 1 + abs\big(f(X_i)\big) & if \;\; f(X_i) < 0 \end{cases} \tag{3.3}$$

such that $f(X_i)$ is the value of the objective function at the i[th] food source X_i.

At the beginning of this phase, three parameters have to be determined: the population size N (the double of the number of food sources SN), the maximum number of cycles or iterations (when this number is reached, the algorithm terminates) and the *limit* (the maximum number of failures that any employed bee can reached before it abandons its food source and converts into a scout bee). Also, the parameter *counter* is initialized to zero for all the food sources in this phase (this parameter records the number of consequent failures of any employed bee and the failure occurs whenever the corresponding solution does not improve).

At the end of this phase, the main loop starts, which contains the employed bee phase, the onlooker bee phase, and the scout bee phase. This loop terminates when the maximum number of iterations is reached. Each iteration is called a *generation*.

3.3.2 Employed Bee Phase

There are as many employed bees as the food sources. Thus, each employed bee searches the neighborhood of its corresponding food source by choosing randomly only one dimension and changing its value according to the following equation:

$$S_{i,j}^t = X_{i,j}^t + r\left(X_{i,j}^t - X_{k,j}^t\right) \;\; for\; i,k \in \{1,\dots,SN\}, j \in \{1,\dots,D\} \tag{3.4}$$

where $S_{i,j}^t$ is the j[th] dimension of the i[th] new solution S_i^t in the neighborhood of the old (parent) i[th] food source X_i^t in the t[th] iteration. k is chosen randomly from the set $\{1,\dots,SN\}$ such that k is different from i, and j is chosen randomly from $\{1,\dots,D\}$. r is a random number such that $r \in [-1,1]$. So the new solution S_i^t is the same as the old solution X_i^t except that the j[th] dimension of X_i^t is changed according to Eq. (3.4) to generate this new

solution S_i^t. If $S_{i,j}^t < L_j$ or $S_{i,j}^t > U_j$, then $S_{i,j}^t$ is set to L_j or U_j, respectively (in this case, $S_{i,j}^t$ violates its predetermined bounds).

Then, the fitness of this new solution $fitness(S_i^t)$ is calculated using Eq. (3.3). if $fitness(S_i^t) \geq fitness(X_i^t)$, the old source food X_i^t is replaced by the new source food S_i^t and *counter(i)* is reset to zero. Otherwise, the old solution X_i^t is kept and *counter(i)* is increased by one as the food source X_i^t does not improve. The aforementioned procedure is repeated for all the food sources.

3.3.3 Onlooker Bee Phase

In the employed bee phase, the quality (i.e., fitness) of SN food sources are calculated by the SN employed bees. In the onlooker bee phase, SN onlooker bees exist, and each one of them will choose one food source to search depending upon some probability. The probability of the i[th] food source to be chosen by any onlooker bee is:

$$p_i = \frac{fitness(X_i^t)}{\sum_{j=1}^{SN} fitness(X_j^t)} \tag{3.5}$$

Once an onlooker bee chooses a food source according the selecting probability of this food source, this onlooker will generate a new solution in the neighborhood of this food source using Eq. (3.4) in a similar way to the employed bees. Like the employed bee phase, if the fitness of the new solution is greater than or equal to the fitness of the old food source, the old food source will be replaced by the new solution and the parameter *counter* of this food source will be reset to zero. Otherwise, the old food source will be kept, and its parameter *counter* will be increased by one since it fails to improve. All the above-mentioned procedure will be repeated for all the SN onlooker bees.

3.3.4 Scout Bee Phase

In this phase, the employed bee whose *counter* value is the maximum among all the employed bees will be chosen. If the *counter* value of this employed bee is not greater than the parameter *limit*, nothing will happen in this phase and the next iteration will start. Nonetheless, if the *counter* value of this employed bee is greater than the parameter *limit*,

then this employed bee will abandon its food source and convert into a scout bee in order to search randomly a new food source using Eq. (3.2). Once the new food source has been found, this scout bee returns to an employed bee and this new food source becomes its corresponding food source. Also, the parameter *counter* of this employed bee will be reset to zero. Then, the next iteration will start.

Note: ABC uses the same equation (i.e., Eq. (3.4)) in both the employed bee and onlooker bee phases. This equation is called the *search equation*.

Figure 3.1 shows the pseudo-code of the ABC algorithm. The termination criterion used in this pseudo-code is the maximum number of iterations *max_iteration*. There are still other termination criteria such as the maximum number of objective function evaluations and acceptable tolerance.

One key drawback of ABC is its slow convergence owing to its poor exploitation ability. There are two main themes to improve the performance of ACB. The first one is achieved by adding some modifications to the original algorithm, like DFSABC, whereas the other theme is achieved by hybridizing the original algorithm with other meta-heuristic algorithms, for example, TLABC.

3.4 ABC with Depth-first-Search Framework (DFSABC)

It is known that ABC has good exploration but, unfortunately, poor exploitation due to its search equation (i.e., Eq. (3.4)), so it has slow convergence [6]. Many modifications have been added to ABC in order to improve its exploitation ability. One of these modified versions is DFSABC [6]. DFSABC attempts to maintain a good trade-off between exploitation and exploration by introducing the depth-first-search framework to ABC. The main modifications of this version are:

1. The search equation Eq. (3.4) is replaced by two new search equations such that the employed bee phase uses one of them, and the onlooker bee phase uses the other. These search equations are claimed to increase the exploitation ability of ABC.

Algorithm 1: The ABC Algorithm

// Initialization Phase

Choose the control parameters (*N*, *max_iteration* and *limit*)

Generate randomly SN food sources using Eq. (3.2)

Find $fitness(X_i^0)$ for i=1:SN using Eq. (3.3)

Set *counter(i)=0* for i=1,…,SN

for (t=1: *max_iteration*) **// main loop**

// Employed Bee Phase

for (i=1:SN)

Generate a new solution S_i^t using Eq. (3.4)

Calculate $fitness(S_i^t)$ using Eq. (3.3)

if $fitness(S_i^t) \geq fitness(X_i^t)$

X_i^t is replaced by S_i^t

Set counter(i) = 0

else

Set counter(i) = counter(i) + 1

end if

end for

Calculate the selecting probability p_i for i=1,…,SN using Eq. (3.5)

// Onlooker Bee Phase

for (j=1:SN) *// SN onlooker bees*

while (true)

choose randomly a food source X_i^t

if ($p_i > rand$) // *rand is a random number between 0 and 1*

Generate a new solution S_j^t using Eq. (3.4)

Calculate $fitness(S_j^t)$ using Eq. (3.3)

if $fitness(S_j^t) \geq fitness(X_i^t)$

X_i^t is replaced by S_j^t

Set counter(i) = 0

else

Set counter(i) = counter(i) + 1

end if

break *// go out the while loop*
end if
end while
end for
// Scout Bee Phase
Pick the food source X_{max}^{t} with the maximum *counter* value.
if(*counter(max)>limit*)
Generate a new food source using Eq. (3.2)
X_{max}^{t} is replaced by this new food source.
Set counter(max) = 0
end if
end for
Return the best solution with the highest fitness value

Figure 3.1 pseudo-code of the ABC algorithm

2. Only the best solutions among the available population (known in DFSABC as elite solutions) are used in the onlooker bee phase. This is also claimed to increase exploitation now that more attention is paid towards the best solutions.

These modifications are aimed to keep a good balance between exploitation and exploration and, accordingly, improve the ABC convergence.

Similar to ABC, DFSABC has four phases: the initialization phase, the employed bee phase, the onlooker bee phase, and finally, the scout bee phase. The initialization phase and scout phase are the same in both ABC and DFSABC. On the other hand, the employed bee phase and the onlooker bee phase of DFSABC are different from those of ABC. As a result, these two phases of DFSABC are explained below.

3.4.1 Employed Bee Phase

Like ABC, there are SN employed bees. At the beginning of the main loop, the best T food sources are chosen (T is usually 10% or 20% of the number of food sources SN) and

these best T food sources are called the *elite solutions*. At the beginning of the employed bee phase, a flag parameter *flag1* is set to 1. Then, for each employed bee, if *flag1* is equal to 1, a random food source X_r^t is chosen from the population and the employed bee generates a new solution S_i^t in its neighborhood according to the following equation:

$$S_{i,j}^t = X_{b,j}^t + r\left(X_{b,j}^t - X_{k,j}^t\right) \tag{3.6}$$

where $S_{i,j}^t$ is the j[th] dimension of the i[th] new solution in the t[th] iteration, $X_{b,j}^t$ is the j[th] dimension of one of the best food sources chosen randomly from the elite solutions, X_k^t is chosen randomly from the current population provided that X_k^t and X_b^t are distinct and both of them are different from the food source X_r^t. Also, j is chosen randomly from $\{1, \dots, D\}$ and r is a random number belonging to [-1,1]. If the new solution S_i^t is better than the old food source X_r^t in terms of the objective function, the old food source X_r^t will be replaced by the new solution S_i^t, *counter(r)* will be reset to 0, and *flag1* is reset to 0 (that means the next employed bee will search also the same food source since *flag1* is not 1). Otherwise, the old food source X_r^t will be maintained, *counter(r)* will be increased by one, and *flag1* is set to 1 (that means the next employed bee will search another randomly-chosen food source since *flag1* is 1).

3.4.2 Onlooker Bee Phase

Similar to ABC, there are SN onlooker bees. Like the employed bee phase, a flag parameter *flag2* is set to 1 at the beginning of this phase. For each onlooker bee, if *flag2* is equal to 1, a random food source X_b^t is chosen from the elite solutions and this onlooker bee will generate a new solution S_i^t in its neighborhood using the following equation:

$$S_{i,j}^t = \frac{1}{2}(X_{b,j}^t + X_{best,j}^t) + r\left(X_{best,j}^t - X_{k,j}^t\right) \tag{3.7}$$

where X_{best}^t is the best solution and X_k^t is chosen randomly from the current population provided that X_k^t, X_{best}^t and X_b^t are distinct food solutions. Also, j is chosen randomly from $\{1, \dots, D\}$ and r is a random number belonging to [-1,1]. If the new solution S_i^t is better than the old food source X_b^t in terms of the objective function, the old food source X_b^t will be replaced by the new solution S_i^t, *counter(b)* will be reset to 0, and *flag2* is reset to 0 (that means the next onlooker bee will also search the same food source since *flag2*

Algorithm 2: The DFSABC Algorithm

// Initialization Phase

Choose the control parameters (*N*, *max_iteration* and *limit*)

Generate randomly SN food sources using Eq. (3.2)

Evaluate $f(X_i^0)$ for i=1:SN // $f(X_i^0)$ *is the objective function value at* X_i^0

Set *counter(i)* to 0 for i=1,…,SN

for (t=1: *max_iteration*) **// main loop**

Choose the best T food sources

// Employed Bee Phase

flag1=1

for (i=1:SN)

if *flag1 == 1*

Choose a random food source X_r^t from the population

end if

Generate a new solution S_i^t using Eq. (3.6)

Evaluate $f(S_i^t)$

if $f(S_i^t) \leq f(X_r^t)$

X_r^t is replaced by S_i^t

Set counter(r) = 0, flag1 =0

else

Set counter(r) = counter(r) + 1, flag1 =1

end if

end for

// Onlooker Bee Phase

flag2 =1

for (i=1:SN) *// SN onlooker bees*

if *flag2 == 1*

Choose a random food source X_b^t from the elite solutions

end if

Generate a new solution S_i^t using Eq. (3.7)

Evaluate $f(S_i^t)$

if $f(S_i^t) \leq f(X_b^t)$

X_b^t is replaced by S_i^t

Set counter(b) = 0, flag2 =0

else

Set counter(b) = counter(b) + 1, flag2 =1

end if

end for

// Scout Bee Phase

Pick the food source X_{max}^t with the maximum *counter* value.

if (*counter(max)>limit*)

X_{max}^t is replaced by a new food source generated using Eq. (3.2).

counter(max) = 0

end if

end for

Return the best solution with the minimum value of the objective function

Figure 3.2 Pseudo-code of DFSABC

is not 1). Otherwise, the old food source X_b^t will be maintained, *counter(b)* will be increased by one, and *flag2* is set to 1 (that means the next onlooker bee will search another randomly chosen food source since *flag2* is 1). Figure 3.2 (above) shows the complete pseudo-code of the DFSABC algorithm.

3.5 Teaching-learning-based Optimization (TLBO) Algorithm

TLBO is inspired by the process of teaching and learning [7]. It mimics the effect of a teacher and how this effect influences his students in terms of their grades. The potential solutions are called *learners* while the learner with the maximum knowledge among all the learners, i.e., the best solution, is modeled as the *teacher*. In each iteration of the algorithm, a population of solutions is produced, and this population is termed as a *class*. The design variables of the optimization problem are modeled as learners' *subjects* while one learner's overall grade is modeled as the *fitness* of that solution, which is inversely proportional to the objective function value. One obvious advantage of TLBO is that there are no any algorithm-specific control parameters and only the general control parameters

(i.e., the population size and the number of iterations) exist. TLBO is mainly comprised of three phases: the *initialization phase*, *teacher phase* and *learner phase*. In the teacher phase, learners learn from the teacher whereas, in the learner phase, the learners learn from each other. Below is a detailed explanation of these three phases.

3.5.1 Initialization Phase

In this phase, N random solutions are generated according to Eq. (3.2) where N is the population size. Also, the control parameters are specified in this phase. The control parameters are the population size *N* and the number of iterations or generations *max_iteration*. At the end of the initialization phase, the main loop starts containing *max_iteration* iterations (i.e., generations), and for each iteration, the following happens:

3.5.2 Teacher Phase

In this phase, the teacher shares his knowledge with the learners in order to improve their own knowledge. First, the best solution is identified and labeled as the teacher X_T^t. Then, for each learner (i.e., solution), a new solution is produced according to the following equation:

$$S_i^t = X_i^t + r(X_T^t - TF \cdot \overline{X^t}) \tag{3.8}$$

Such that

$$TF = round(r + 1) \tag{3.9}$$

where S_i^t is the ith new solution, X_T^t is the best solution (i.e., the teacher) in the tth iteration, X_i^t is the ith learner in the tth iteration, $\overline{X^t}$ is the mean of the class (i.e., the mean of the current population of solutions), TF is called the *teaching factor* and it is either 1 or 2 according to the above formula, and r is a random number between 0 and 1. Then, if the new solution S_i^t is better than the old solution X_i^t (i.e., the objective function at S_i^t is less than its value at X_i^t), the old solution X_i^t will be replaced by the new one S_i^t and it is said that the ith learner learns something new. Otherwise, the old solution will be maintained.

3.5.3 Learner Phase

In this phase, the learners share their knowledge with each other to improve their own knowledge. For each learner X_i^t, another learner X_k^t is chosen randomly to interact with this learner. Then, a new solution S_i^t is generated according to:

$$S_i^t = \begin{cases} X_i^t + \mathrm{r}(X_i^t - X_k^t) & if\ f(X_i^t) < f(X_k^t) \\ X_i^t + \mathrm{r}(X_k^t - X_i^t) & if\ f(X_i^t) \geq f(X_k^t) \end{cases} \quad (3.10)$$

where $f(X_i^t)$ is the value of the objective function at the i[th] learner X_i^t and $f(X_k^t)$ is the value of the objective function at the randomly selected learner X_k^t (i and k are different). Similar to the teacher phase, the new solution S_i^t will replace the old solution X_i^t only if the new solution S_i^t is better than the old solution X_i^t (i.e., $f(S_i^t) < f(X_i^t)$). Figure 3.3 shows the pseudo-code of TLBO.

3.6 Teaching-learning-based ABC (TLABC)

TLABC is a hybridization of the ABC and TLBO algorithms in order to improve the convergence and performance of the ABC algorithm. This hybrid algorithm makes use of the high exploitation ability of TLBO and high exploration ability of ABC to generate a new algorithm with a reasonable balance between exploitation and exploration. The TLABC algorithm comprises four phases: the initialization phase, the teaching-based employed bee phase, the learning-based onlooker bee phase, and lastly, the generalized oppositional scout bee phase. The initialization phase is the same as the initialization phase of the original ABC algorithm. Therefore, only the other three phases are explained below.

3.6.1 Teaching-based Employed Bee Phase

Each employed bee searches around its corresponding food source (i.e., solution) and generates a new solution using the following equation:

$$S_{i,j}^t = \begin{cases} X_{i,j}^t + r_2\left(X_{T,j}^t - TF \cdot \overline{X_j^t}\right) & if\ r_1 < 0.5 \\ X_{k,j}^t + a\left(X_{h,j}^t - X_{l,j}^t\right) & otherwise \end{cases} \quad (3.11)$$

Algorithm 3: The TLBO Algorithm

// Initialization Phase

Choose the control parameters (*N*, *max_iteration*)

Generate randomly N learners (i.e., solutions) using Eq. (3.2)

Evaluate $f(X_i^0)$ for i=1:N

for (t=1: *max_iteration*) // **main loop**

// Teacher Phase

for (i=1:N)

Identify the best solution (i.e., Teacher) X_T^t

Calculate the population mean $\overline{X^t}$

Generate a new solution S_i^t using Eq. (3.8)

if $f(S_i^t) < f(X_i^t)$

X_i^t is replaced by S_i^t

end if

end for

// Learner Phase

for (j=1:N)

Choose randomly another learner X_k^t

Generate a new solution S_i^t using Eq. (3.10)

if $f(S_i^t) < f(X_i^t)$

X_i^t is replaced by S_i^t

end if

end for

end for

Return the best solution with the minimum value of the objective function

Figure 3.3 Pseudo-code of TLBO

where $S_{i,j}^t$ is the j[th] dimension of the i[th] new solution in the t[th] iteration, $X_{i,j}^t$ is the j[th] dimension of the i[th] food source (the i[th] old solution), $X_{T,j}^t$ is the j[th] dimension of the teacher (i.e., the current best solution), $\overline{X_j^t}$ is the j[th] dimension of the population mean, X_k^t, X_h^t and X_l^t are three distinct food sources and they are also different from the i[th] food

source, r_1 and r_2 are random numbers between 0 and 1, TF is the *teaching factor* (similar to TLBO), and a is a *scale factor* belonging to [0,1]. Now, if the new solution S_i^t is better than the old one X_i^t, the old solution X_i^t will be replaced by S_i^t and *counter(i)* will be reset to zero. Otherwise, the old solution will be maintained and *counter(i)* will be increased by one.

3.6.2 Learning-based Onlooker Bee Phase

Each onlooker bee chooses one food source depending on the selecting probability of that food source p_i (this probability is calculated using Eq. (3.5)). Once the onlooker bee has chosen a food source X_i^t, it produces a new solution in the neighborhood of this food source according to this equation:

$$S_i^t = \begin{cases} X_i^t + r(X_i^t - X_k^t) & if\ f(X_i^t) \leq f(X_k^t) \\ X_i^t + r(X_k^t - X_i^t) & if\ f(X_i^t) > f(X_k^t) \end{cases} \quad (3.12)$$

where X_k^t is a food source randomly chosen from the current population and different from the i[th] food source X_i^t. As always, r is a random number between 0 and 1. If the new solution S_i^t is better than the old one X_i^t, then S_i^t replaces X_i^t and *counter(i)* will be reset to zero. Otherwise, the old solution will be maintained and *counter(i)* will be increased by one.

3.6.3 Generalized Oppositional Scout Bee Phase

In this phase, the employed bee whose *counter* value is the maximum among all the employed bees will be chosen. If the *counter* value of this employed bee is greater than the parameter *limit*, then this employed bee will abandon its food source and converts into a scout bee. This scout bee will generate a new random solution X_i using Eq. (3.2) and its generalized oppositional solution X_o as well. The generalized oppositional solution is produced using the following equation:

$$X_{o,j} = r\left(a_j + b_j\right) - X_{i,j} \quad for\ j = 1, \dots, D \quad (3.13)$$

where $X_{o,j}$ is the j[th] dimension of the generalized oppositional solution X_o and r is a random number between 0 and 1. Also, a_j and b_j are the maximum and minimum values of the j[th] dimension among the current population.

Then, one of these two solutions with the higher fitness will replace the old abandoned food source. Once the new food source has been found, this scout bee returns to an employed bee and this new food source becomes its corresponding food source. Also, the parameter *counter* of this employed bee will be reset to zero. Figure 3.4 shows the pseudo-code of TLABC.

3.7 Cuckoo Search (CS)

CS is another nature-inspired meta-heuristic optimization algorithm. It mimics the intelligent breeding behavior of particular cuckoo species where they search for host nests to lay their eggs [8]. In CS, every potential solution is represented as a *cuckoo egg*, or a *nest* because every nest has only one egg, and each cuckoo searches randomly for a potential nest to lay its egg (i.e., new solution). Like other meta-heuristic algorithms, CS starts by initializing N random solutions (i.e., nests) using Eq. 3.2. Therefore, there are N cuckoos with N corresponding nests. Apart from the initialization phase, CS mainly consists of two phases: the *local random walk phase* and the *global random walk phase*.

3.7.1 Global Random Walk Phase

In this phase, each cuckoo generates a new solution S_i^t using this equation:

$$S_i^t = X_i^t + \alpha_1 \cdot L(\lambda) \tag{3.14}$$

such that the Levy-flights-based step size $L(\lambda)$ is distributed as follows

$$L(\lambda) \sim \frac{\lambda \Gamma(\lambda) \sin(\pi\lambda/2)}{\pi} \cdot \frac{1}{\delta^{1+\lambda}} \tag{3.15}$$

where X_i^t is the i[th] cuckoo's old nest (old solution) in the t[th] iteration, $\alpha_1 > 0$ is the *step size* (often $\alpha_1 = 0.01$) and the Levy exponent λ is normally set as λ =1.5. Then, if the new solution S_i^t is better than the cuckoo's old nest X_i^t, X_i^t will be replaced by S_i^t.

Algorithm 4: The TLABC Algorithm

// Initialization Phase

Choose the control parameters (*N*, *max_iteration* and *limit*)

Generate randomly SN food sources using Eq. (3.2)

Find $fitness(X_i^0)$ for i=1:SN using Eq. (3.3)

Set *counter(i)=0* for i=1,…,SN

for (t=1: *max_iteration*) // **main loop**

// Teaching-based Employed Bee Phase

for (i=1:SN)

Generate a new solution S_i^t using Eq. (3.11)

Calculate $fitness(S_i^t)$ using Eq. (3.3)

if $fitness(S_i^t) \geq fitness(X_i^t)$

X_i^t is replaced by S_i^t

Set counter(i) = 0

else

Set counter(i) = counter(i) + 1

end if

end for

Calculate the selecting probability p_i for i=1:SN using Eq. (3.5)

// Learning-based Onlooker Bee Phase

for (j=1:SN)

Pick a food source X_i^t using the roulette method based on p_i

Generate a new solution S_i^t using Eq. (3.12)

Calculate $fitness(S_i^t)$ using Eq. (3.3)

if $fitness(S_i^t) \geq fitness(X_i^t)$

X_i^t is replaced by S_i^t

Set counter(i) = 0

else

Set counter(i) = counter(i) + 1

end if

end for

// Scout Bee Phase

Pick the food source X_{max}^{t} with the maximum *counter* value.
if(*counter(max)>limit*)
Generate a new food source X_i using Eq. (3.2)
Generate its generalized oppositional solution X_o using Eq. (3.13)
if ($fitness(X_i) \geq fitness(X_o)$)
X_{max}^{t} is replaced by X_i
else
X_{max}^{t} is replaced by X_o
Set counter(max) = 0
end if
end for
Return the best solution with the highest fitness value

Figure 3.4 Pseudo-code of TLABC

3.7.2 Local Random Walk Phase

In this phase, each cuckoo generates a new solution S_i^t using this equation:

$$S_i^t = X_i^t + \alpha_2 \otimes H(p_a - r) \otimes \left(X_j^t - X_k^t\right) \quad (3.16)$$

where X_i^t is the i[th] cuckoo's old nest (old solution) in the t[th] iteration, $\alpha_2 > 0$ is the *step size* (it is drawn from a uniform distribution in this phase), H(u) is the Heaviside function, X_j^t and X_k^t are two randomly-selected distinct solutions, the switch probability p_a is the fraction of the worst nests to be replaced by new random solutions (p_a is normally set to be 0.25), and lastly, r is a random number belonging to [0,1]. In the preceding equation, the symbol $\otimes$ refers to the element-wise multiplication of vectors. Then, if the new solution S_i^t is better than the cuckoo's old nest X_i^t, X_i^t will be replaced by S_i^t. Finally, Figure 3.5 depicts the pseudo-code of the CS algorithm.

Algorithm 5: The CS Algorithm

// Initialization Phase

Choose the control parameters (*N*, *max_iteration* and p_a)

Generate randomly N host nests using Eq. (3.2)

Evaluate $f(X_i^0)$ for i=1:N

// main loop

for (t=1: *max_iteration*)

// Global Random Walk Phase

for (i=1:N)

Generate a new nest S_i^t by Levy Flights using Eq. (3.14)

Evaluate $f(S_i^t)$

if $f(S_i^t) < f(X_i^t)$

X_i^t is replaced by S_i^t

end if

end for

// Local Random Walk Phase

p_a of the worst solutions (i.e., solutions) are abandoned in this phase

for (i=1:N)

Generate a new nest S_i^t using Eq. (3.16)

Evaluate $f(S_i^t)$

if $f(S_i^t) < f(X_i^t)$

X_i^t is replaced by S_i^t

end if

end for

Determine the current best solution

end for

Return the best solution with the minimum value of the objective function

Figure 3.5 Pseudo-code of CS

CHAPTER 4

METHODOLOGY

This work aims to improve the performance of SVMs by tuning their hyper-parameters. In fact, these hyper-parameters are tuned herein according to the wheat varieties dataset. Firstly, the flowchart of the proposed methodology is presented and explained in the first section. Then, the following sections shed light on the details of this flowchart, such as the formation of the optimization problem and the SVM classifier.

4.1 Overview of Methodology

First, Figure 4.1 depicts the main system for the classification process of the wheat varieties dataset. At the beginning, the data have to be preprocessed. Then, the SVM classifier is used to classify these data into one of their wheat varieties: Canadian, Kama and Rosa. However, SVMs involve some hyper-parameters whose values must be chosen appropriately prior to training and testing. The meta-heuristic optimization algorithms, namely ABC, DFSABC, TLBO, TLABC and CS, search for the optimal values of these hyper-parameters.

In addition, Figure 4.2 shows the proposed methodology as a detailed flowchart. As mentioned in the previous chapter, the meta-heuristic optimization algorithms contain some control parameters that should be determined beforehand. Therefore, at the beginning of the flowchart, the proper values of these control parameters have to be specified. This step is considered as an initialization step, just like the data preprocessing step. In the main part of the flowchart, the meta-heuristic algorithms with their proper control parameters search for the optimal SVM hyper-parameters that result in the minimum five-fold cross-validation error and then the maximum accuracy accordingly. In the main loop of the flowchart, these hyper-parameters are used with the SVM classifier to classify the wheat varieties dataset, and the obtained five-fold cross-

validation error is fed back to the meta-heuristic algorithms as the value of the objective function in order to update their search and solutions. The same process is repeated until the termination criterion, which is the maximum number of objective function evaluations in this work, is reached. As soon as the termination criterion has been reached, the flowchart ends and outputs the best classification accuracy, which is corresponding to the minimum five-fold Cross-validation error.

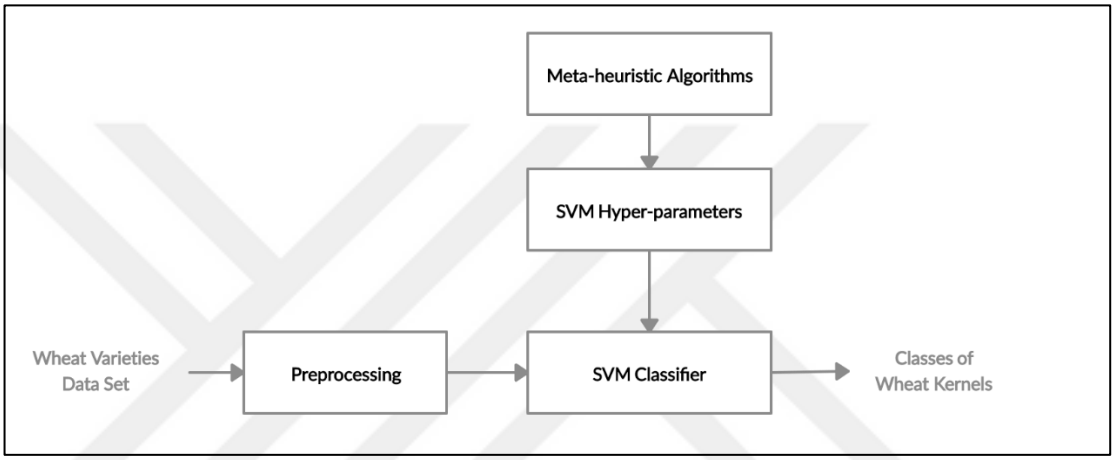

Figure 4.1 Block diagram of the classification system of wheat kernels

In general, the main work of this thesis is divided into two major stages.

1. **First stage:** the experiments are performed to determine the appropriate general control parameters of the meta-heuristic algorithms. These general control parameters, which are common for all the meta-heuristic algorithms, are the population size N and the maximum number of objective function evaluations. Four population sizes are tested; these population sizes are 20, 30, 50 and 100. Moreover, the maximum number of objective function evaluations, which is defined as the termination criterion, is set to 10,000 evaluations in the first stage. This stage tries to discover how many enough function evaluations are required to achieve acceptable results. Regarding the algorithm-specific control parameters, whose values differ from one algorithm to another, their values are selected according to what is recommended in the literature.

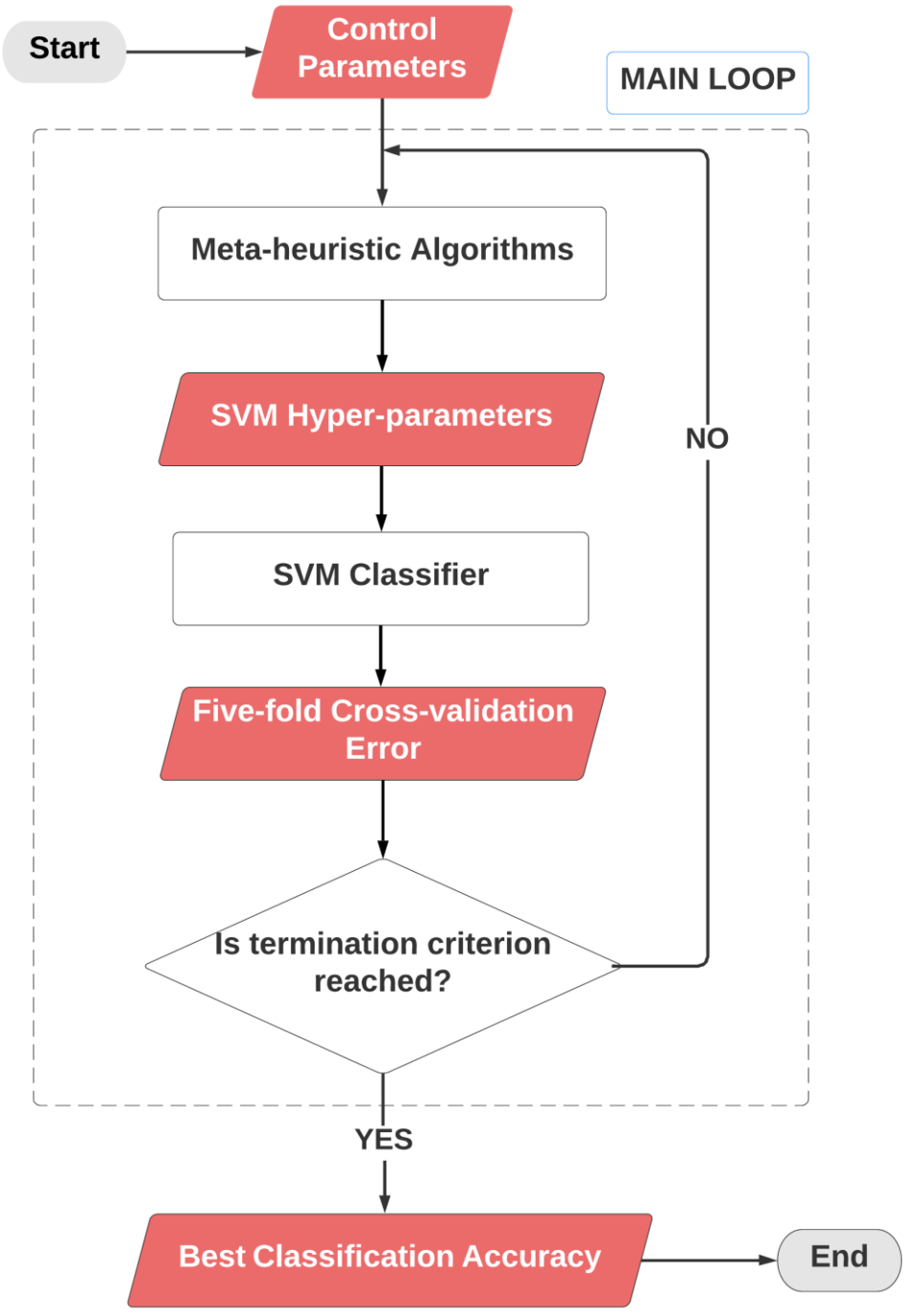

Figure 4.2 flowchart of the methodology

The algorithm-specific control parameters of the five meta-heuristic optimization algorithms used in this work are shown in Table 4.1. The value of the parameter *limit* in ABC, DFSABC and TLABC are chosen as $SN \cdot D$ where SN is the number of food sources and D is the dimensionality of the optimization problem.

This value of the parameter *limit* is recommended in [4]. The number of elite solutions in DFSABC is selected as 20% of the number of available food sources according to the guidelines in [6]. The parameter p_a of CS, which represents the fraction of worst solutions to be replaced in each generation, is selected to be 0.25 as recommended in [8]. Unlike the other four meta-heuristic algorithms, TLBO does not have any algorithm-specific control parameters, which is deemed an advantage over the other algorithms.

Table 4.1 Algorithm-specific control parameters of meta-heuristic algorithms

Meta-heuristic Algorithm	**Algorithm-specific Control Parameters**	**Values**
ABC	*limit*	$SN \cdot D$
DFSABC	*limit*	$SN \cdot D$
	Number of elite solutions	$0.2 \cdot SN$
TLBO	-	-
TLABC	*limit*	$SN \cdot D$
CS	p_a	0.25

2. **Second stage:** the five meta-heuristic algorithms, with the appropriate control parameters chosen in the first stage, tune the SVM hyper-parameters according to the wheat varieties dataset as shown in the flowchart in Figure 4.2. These algorithms attempt to find the optimal values of the hyper-parameters which result in the minimum five-fold cross-validation error. With these optimal values of the hyper-parameters, the SVM classifier, therefore, classifies the wheat varieties dataset with the maximum possible accuracy. Ultimately, the results of the five meta-heuristic algorithms are statistically compared against each other in terms of performance and computational time. Likewise, a statistical comparison is drawn between the four types of SVMs using the same measures. The subsequent sections explain in detail the contents of the flowchart shown in Figure 4.2.

4.2 Wheat Varieties Dataset

The wheat varieties dataset is also known as the *seeds* dataset in the UCI Machine Learning Repository [33]. This dataset consists of 210 X-ray images of wheat kernels. These kernels belong to three distinct wheat classes or varieties, namely Rosa, Kama and Canadian, each of which is comprised of 70 images. The images were taken using soft X-ray photography. Then, the morphological features concerning the physical structure of the kernels were extracted from these images [10]. These morphological features are seven geometric features:

1. *Area* A (measured in [mm^2])
2. *Perimeter* P (measure in [m])
3. *Length of kernel* (measured in [mm])
4. *Width of kernel* (measured in [mm])
5. *Length of kernel groove* (measured in [mm])
6. *Compactness* C = $\frac{4\pi A}{P^2}$ (measured as a percentage between zero and one)
7. *Asymmetry coefficient*, which is defined as a ratio of two quantities such that the numerator quantity is the absolute difference between the area of the right part and the area of the left part, whereas the denominator quantity is the area of the kernel A [85].

Table 4.2 shows the descriptive statistics of these seven geometric features. It displays the minimum, maximum, average, and standard deviation of the features for each class of the three classes and the total data as well. It is apparent from the table that different features have different scales and ranges. Consequently, there is a considerable need for data preprocessing and normalization.

It is always beneficial to visualize the data to understand how the data and their three classes are distributed across their space. Unfortunately, the data cannot be visualized since the dimensionality of the data is seven (i.e., the data consist of seven features or attributes), which means the data reside in a 7-dimensional space.

Table 4.2 Descriptive statistics of the dataset

Geometric Features		**Min**	**Max**	**Mean**	**Std dev.**
Area	Kama	11.23	17.08	14.33	1.216
	Rosa	15.38	21.18	18.33	1.439
	Canadian	10.59	13.37	11.87	0.723
	Total	10.59	21.18	14.85	2.91
Perimeter	Kama	12.63	15.46	14.29	0.5766
	Rosa	14.66	17.25	16.14	0.617
	Canadian	12.41	13.95	13.25	0.3402
	Total	12.41	17.25	14.56	1.306
kernel Length	Kama	4.902	6.053	5.508	0.2315
	Rosa	5.363	6.675	6.148	0.2682
	Canadian	4.899	5.541	5.23	0.138
	Total	4.899	6.675	5.629	0.4431
kernel Width	Kama	2.85	3.683	3.245	0.1776
	Rosa	3.231	4.033	3.677	0.1855
	Canadian	2.63	3.232	2.854	0.1475
	Total	2.63	4.033	3.259	0.3777
kernel groove Length	Kama	4.519	5.877	5.087	0.2637
	Rosa	5.144	6.55	6.021	0.2539
	Canadian	4.745	5.491	5.116	0.1621
	Total	4.519	6.55	5.408	0.4915
Compactness	Kama	0.8392	0.9183	0.8801	0.01619
	Rosa	0.8452	0.9108	0.8835	0.0155
	Canadian	0.8081	0.8977	0.8494	0.02176
	Total	0.8081	0.9183	0.871	0.02363
Asymmetry coefficient	Kama	0.7651	6.685	2.667	1.174
	Rosa	1.472	6.682	3.645	1.182
	Canadian	1.661	8.456	4.788	1.336
	Total	0.7651	8.456	3.7	1.504

However, a dimensionality reduction algorithm, fortunately, can be implemented in order to reduce the dimensionality to two or three, so the data can be drawn in 2-dimensional or 3-dimensional spaces, respectively. The principal component analysis (PCA) algorithm [86] is widely used for dimensionality reduction. PCA is run here to reduce the dimensionality of the data from seven to two, and then the data can be displayed in two-dimensional space, as shown in Figure 4.3.

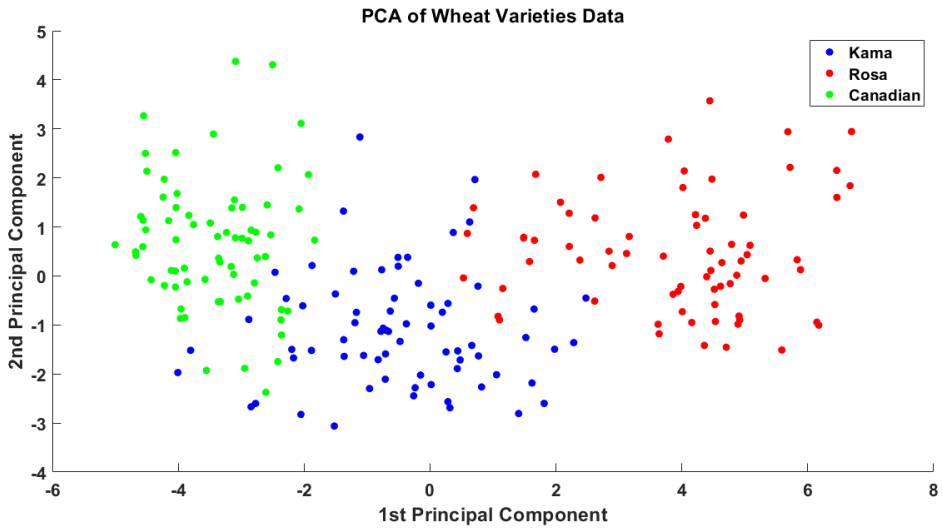

Figure 4.3 Displaying the data in 2D using PCA

The main notion of PCA is that not all the original axes (i.e., original features) are independent variables; instead, they are inter-correlated to some extent and some of them might be dependent upon the others. Therefore, PCA transforms the data from their original space with the original axes (i.e., attributes of the data) into a new space with new axes (i.e., new attributes/features). These new axes are called *principal components* and they are nothing but a linear weighted combination of the original axes. In addition, these principal components are ordered according to their significance and independence such that only the first principal components matter. In other words, the first component is the most significant and most independent variable, while the last component is the least significant and most likely to be a dependent variable. In Figure 4.3, the data are represented using only the first two principal components of the PCA algorithm. In fact, these two principal components maintain and explain approximately 99.3% of the variability (i.e., information) of the original data. That is, these two principal components

are the major independent axes that can represent the original data sufficiently, whereas the remaining principal components are insignificant enough to be neglected. Figure 4.3 also shows the data of each one of the three wheat classes and reveals their distributions across the principal component space. In this space, the three classes are not completely distinct from each other and there are some overlapped data points. That reveals the fact that the data are non-separable, and they require an efficient learning algorithm to classify them accurately.

4.3 Preprocessing

The data are needed first to be prepared before using it with the SVM classifier. Normally, the data are shuffled and normalized. Normalizing the data is important because different features of the data normally have different ranges and scales. Nevertheless, all the features should have the same scale, so they have equal weight during training and testing. Once all the features have the same weight, it is ensured that no one specific feature will dominate during training. In addition, normalizing data is known to increase the learning speed in many ML algorithms. In fact, in [87], it is stated that normalizing the data improves the SVM performance.

As a result, in this work, the data are normalized using the *min-max normalization* in order to have all the features (i.e., the seven variables or attributes of the dataset) fall into the same range [-1,1]. This normalization is done using the following equation:

$$z_i = \frac{2\,[\,x_i - \min(x_i)\,]}{\max(x_i) - min\,(x_i)} - 1 \qquad for\ i = 1, \dots, 7 \tag{4.1}$$

where x_i is the ith feature and $z_i \in [-1,1]$ is the normalized ith feature.

4.4 SVM Classifier

SVMs are used here to classify the wheat kernels into three varieties: Kama, Rosa and Canadian. Since this problem has three distinct classes, this is a multiclass classification problem. Accordingly, the one-versus-one scheme is used with SVMs to solve this multiclass classification problem (more details about the one-versus-one scheme are

explained in Section 2.5). In this work, four types of SVMs are implemented, each of which uses a particular kernel function. These four types are:

1. **Gaussian SVMs:** use the gaussian kernel function.
2. **Polynomial SVMs:** use the polynomial kernel function.
3. **Sigmoid SVMs:** use the sigmoid kernel function.
4. **Hybrid-kernel SVMs:** use a hybrid kernel function which is defined as a weighted sum of the Gaussian and polynomial kernel functions.

Their mathematical formulae and descriptions were presented in Section 2.3. However, each algorithm of these four algorithms has its hyper-parameters whose values affect the overall performance. Therefore, the values of these hyper-parameters have to be tuned according to the wheat varieties dataset so as to achieve the best possible accuracy. The subsequent section describes this tuning process which can be deemed an optimization problem.

4.5 Optimization Problem

The main optimization problem of this work can be formed as:

$$\min_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) \tag{4.2}$$

where $f(\boldsymbol{\theta})$ is the objective function that is needed to be minimized and $\boldsymbol{\theta}$ is a vector containing the design variables with respect to which $f(\boldsymbol{\theta})$ is minimized. Then, an efficient optimization algorithm is required to solve this problem. More details about the objective function, design variables and optimization algorithms are presented below.

4.5.1 The Objective Function

In this optimization problem, the five-fold cross-validation error is defined as the objective function that is needed to be minimized. Since the dataset is small (its size is only 210 examples/observations), the five-fold cross-validation error is a more reasonable estimate of the generalization error than the holdout test error. In the holdout testing scheme, one portion of the data is used for testing (usually about one-third of the data) and the remaining data are used for training. The five-fold cross-validation, in fact, is one

type of the general k-fold cross-validation. The k-fold cross validation means that the data are divided randomly into k folds or groups with roughly equal sizes. Then the learning model (i.e., the SVM model in this case) will be trained and tested k times. At the first time, the first fold or group will be set aside for testing and the model will be trained using the remaining (k-1) folds. Similarly, at the second time, the second fold will be kept for testing while the model will be trained using the remaining (k-1) folds. In a similar way, the same procedure will be followed for all the remaining folds such that, at each time, one fold is used for testing and the remaining folds will be used for training. Also, the testing error will be saved at each time. Then, the overall cross-validation error is the average testing error (i.e., the average of the k testing errors) [88]. This error is the misclassification rate, which is defined as the proportion of the testing data that are misclassified. The most widely used k-fold cross-validation schemes are the five-fold and ten-fold cross-validation schemes. In this work, the five-fold cross-validation is chosen instead of the ten-fold cross-validation since the computational burden increases as k increases and, accordingly, the five-fold cross-validation is less computationally expensive. Furthermore, the five-fold cross-validation is more frequent with small datasets to ensure that the testing fold at each time is not too small. If the testing fold is too small, it does not appropriately represent the data and then it biases the performance [89].

4.5.2 Design Variables

The design variables of this optimization problem are the SVM hyper-parameters. These hyper-parameters are the trade-off parameter C as well as the kernel parameters. As described in Section 2.3, many different kernel functions exist, each of which has its own kernel parameters. Accordingly, in this work, the meta-heuristic algorithms optimize four types of SVMs, each of which has its different kernel function. These four types are Gaussian SVMs, polynomial SVMs, sigmoid SVMs and hybrid-kernel SVMs. Table 4.3 depicts the kernel parameters of each kernel function along with their search ranges used in this study. Concerning the trade-off parameter C, all the four aforementioned SVMs share this hyper-parameter C besides their own kernel parameters.

Table 4.3 Four different variants of SVMs using four kernel functions

SVM Type	Mathematical Formula	Kernel Parameters	Search Ranges of Parameters
Gaussian SVMs	$exp(-\frac{1}{2\sigma}\|\boldsymbol{x_i}-\boldsymbol{x_j}\|^2)$	Kernel scale σ	$[2^{-14},2^{14}]$
Polynomial SVMs	$(\boldsymbol{x_i}\cdot\boldsymbol{x_j}+1)^r$	Polynomial degree r	[1,10]
Sigmoid SVMs	$\tanh(\gamma\boldsymbol{x_i}\cdot\boldsymbol{x_j}-\delta)$	Kernel scale γ Offset parameter δ	$[2^{-14}, 2^{14}]$ [0,5]
Hybrid-kernel SVMs	$\beta\cdot exp\left(-\frac{1}{2\sigma}\|\boldsymbol{x_i}-\boldsymbol{x_j}\|^2\right)+(1-\beta)\cdot(\boldsymbol{x_i}\cdot\boldsymbol{x_j}+1)^r$	Kernel scale σ Polynomial degree r Weight β	$[2^{-14}, 2^{14}]$ [1,10] [0,1]

Note: As regards the positive real hyper-parameters with wide ranges (i.e., the trade-off parameter C, Gaussian kernel scale σ and sigmoid kernel scale γ), the optimization algorithms search for their logarithms instead of their direct values. That is, the optimization algorithms search for $\log_2 C$, $\log_2 \sigma$ and $\log_2 \gamma$. This procedure makes the search ranges symmetric and narrower, not to mention the fact that the optimization problem becomes more stable [42]. In this work, the search ranges of $\log_2 C$, $\log_2 \sigma$ and $\log_2 \gamma$ are set as [-14, 14] and the ranges of C, σ and γ become $[2^{-14},2^{14}]$, then.

Therefore, the design variables (i.e., optimization parameters) $\boldsymbol{\theta}$ in the optimization problem, $\min_{\boldsymbol{\theta}} f(\boldsymbol{\theta})$, are defined in the four types of SVMs as follows:

1. **Gaussian SVMs:** the design variables $\boldsymbol{\theta} = (\theta_1, \theta_2)$ where θ_1 is equal to $\log_2 \sigma$ and θ_2 is equal to $\log_2 C$. Regarding their ranges, $\theta_1 \in [-14,14]$ and $\theta_2 \in [-14,14]$. Hence, the search space of the Gaussian SVM hyper-parameter optimization is two-dimensional such that there exist only two hyper-parameters.
2. **Polynomial SVMs:** the design variables $\boldsymbol{\theta} = (\theta_1, \theta_2)$ where θ_1 is equal to r and θ_2 is equal to $\log_2 C$. Regarding their ranges, $\theta_1 \in \{1,2,\dots,10\}$ and $\theta_2 \in [-14,14]$. Hence, the search space of the polynomial SVM hyper-parameter optimization is also two-dimensional.

3. **Sigmoid SVMs:** the design variables $\boldsymbol{\theta} = (\theta_1, \theta_2, \theta_3)$ where θ_1 is equal to $\log_2 \gamma$, θ_2 is equal to $\log_2 C$ and θ_3 is equal to δ. Regarding their ranges, $\theta_1 \in [-14,14]$, $\theta_2 \in [-14,14]$ and $\theta_3 \in [0,5]$. Hence, the search space of the sigmoid SVM hyper-parameter optimization is three-dimensional such that there exist three hyper-parameters.
4. **Hybrid-kernel SVMs:** the design variables $\boldsymbol{\theta} = (\theta_1, \theta_2, \theta_3, \theta_4)$ where $\theta_1$is equal to $\log_2 \sigma$, θ_2 is equal to $\log_2 C$, $\theta_3$is equal to r and θ_4 is equal to β. Regarding their ranges, $\theta_1 \in [-14,14]$, $\theta_2 \in [-14,14]$ and $\theta_3 \in \{1,2, \dots ,10\}$ and $\theta_4 \in [0,1]$. Hence, the search space of the hybrid-kernel SVM hyper-parameter optimization is four-dimensional such that there exist four hyper-parameters.

4.5.3 Optimization Algorithms

Five meta-heuristic optimization algorithms are used to optimize the hyper-parameters of the four SVM types: Gaussian SVMs, polynomial SVMs, sigmoid SVMs and hybrid-kernel SVM. These five meta-heuristic optimization algorithms, as mentioned earlier in this chapter, are ABC, DFSABC, TLBO, TLABC, and lastly, CS. More details about these algorithms and their pseudo-code are presented in Chapter 3.

CHAPTER 5
EXPERIMENTAL RESULTS AND DISCUSSION

The experimental work of this thesis is divided into two stages of experiments. In the first stage, the appropriate control parameters of meta-heuristic optimization algorithms are determined. In the second stage, the results of optimizing the four SVM classifiers using these meta-heuristic algorithms with their appropriate control parameters are compared. These results are compared against each other to figure out which SVM classifier is the best and which meta-heuristic algorithm is the best in terms of performance and time consumption.

5.1 First Stage: Searching for the appropriate control parameters

As described in the previous chapter, this work uses four types of SVMs, namely Gaussian, polynomial, sigmoid and hybrid-kernel SVMs. In this stage of the work, the appropriate population size, as well as the sufficient number of required objective function evaluations, are empirically determined. To find these appropriate control parameters of the meta-heuristic algorithms, experiments are run with four population sizes (i.e., 20, 30, 50 and 100) in this stage while the maximum number of objective function evaluations is set as 10,000. Then, from the convergence curves of the algorithms, one can deduce the appropriate population size and the enough number of objective function evaluations. Due to the apparent fact that meta-heuristic algorithms involve randomness affecting their results, each meta-heuristic algorithm is run 30 times for each one of the four SVM classifiers, and the reported results in the coming subsections are the averages of these 30 runs.

5.1.1 Gaussian SVMs

Five meta-heuristic algorithms, namely ABC, DFSABC, TLBO, TLABC and CS, are applied to Gaussian SVMs. Figure 5.1 shows the convergence curves of these five meta-heuristic algorithms while the population size takes four different values: 20 (in blue), 30 (in red), 50 (in yellow), 100 (in purple), and the number of objective function evaluations is equal to 10,000. The convergence curves show how the objective function (i.e., the five-fold cross-validation error) decreases with respect to the number of evaluations. As demonstrated from the curves, there is no significant difference between the four population sizes and their convergence curves look almost the same. Consequently, the population size is chosen as 20 for Gaussian SVMs in the second stage. In addition, it is obvious from Figure 5.1 that around 3,000 evaluations are enough to achieve acceptable results as the meta-heuristic algorithms show no considerable improvement after 3,000 evaluations. It is also well-known that the computational time is proportional to the number of objective function evaluations and a decrease in the latter causes a corresponding decrease in the former. Therefore, it is desirable to reduce the number of objective function evaluations as long as the performance is still acceptable. Thus, the maximum number of objective function evaluations can be selected as 3,000 evaluations without any substantial discrepancy in the results.

5.1.2 Polynomial SVMs

Similar to Gaussian SVMs, the same five meta-heuristic algorithms are applied to polynomial SVMs as well. Figure 5.2 displays the convergence curves of the meta-heuristic algorithms with population sizes: 20 (in blue), 30 (in red), 50 (in yellow), 100 (in purple). For all the five meta-heuristic algorithms, their curves demonstrate that the difference between the four population sizes is insignificant and they look almost the same. Consequently, the population size is chosen as 20 for polynomial SVMs, as well, in the second stage. As regards the number of objective function evaluations, it is apparent that around 3,000 evaluations are enough to achieve acceptable results.

5.1.3 Sigmoid SVMs

Here, the five meta-heuristic algorithms are applied to sigmoid SVMs and Figure 5.3 displays their convergence curves with population sizes: 20 (in blue), 30 (in red), 50 (in yellow), 100 (in purple). For all the five meta-heuristic algorithms, the convergence curves are similar to each other and that demonstrates the absence of substantial discrepancy between the four population sizes. Consequently, the population size is chosen as 20 for sigmoid SVMs, as well, in the second stage. Also, around 3,000 evaluations are sufficient to ensure acceptable results.

5.1.4 Hybrid-kernel SVMs

In the final experiment of the first stage, the five meta-heuristic algorithms are applied to hybrid-kernel SVMs. Figure 5.4 shows their convergence curves with the four population sizes: 20 (in blue), 30 (in red), 50 (in yellow) and 100 (in purple). For all the five meta-heuristic algorithms, their curves demonstrate that the difference between the four population sizes is insignificant and they look almost the same. Consequently, the population size is chosen as 20 for hybrid-kernel SVMs, as well, in the second stage. As regards the number of objective function evaluations, it is apparent that around 3,000 evaluations are enough to achieve acceptable results.

5.2 Second Stage: Optimizing SVM classifiers

In the first stage, the control parameters of the meta-heuristic algorithms are determined. The appropriate population size is chosen as 20 for the five meta-heuristic algorithms and the maximum number of objective function evaluations is selected as 3,000 evaluations. The second stage contains the main work of this thesis. The five meta-heuristic algorithms with those control parameters are implemented to optimize the SVM classifiers. Their results are statistically compared according to their performance and computational time. First, the convergence curves of the meta-heuristic algorithms for each SVM classifier are presented and compared. Then, the performances of these SVM classifiers will be compared against each other. Like the first stage, each meta-heuristic algorithm in this stage is run 30 times and the reported results below are the averages of these 30 runs.

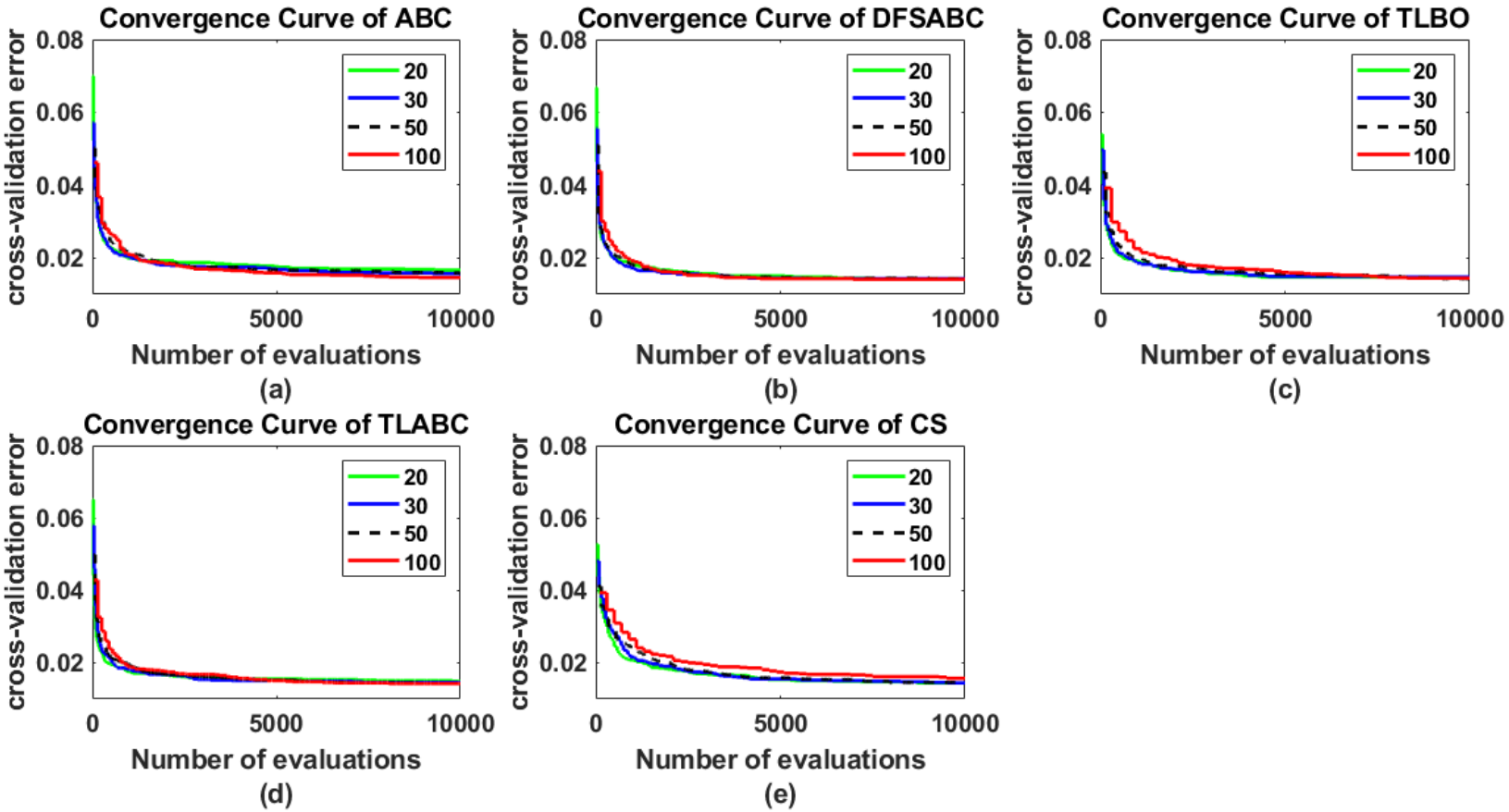

Figure 5.1 Convergence curves of meta-heuristic algorithms for Gaussian SVMs (first stage)
(a) ABC, (b) DFSABC, (c) TLBO, (d) TLABC, (e) CS

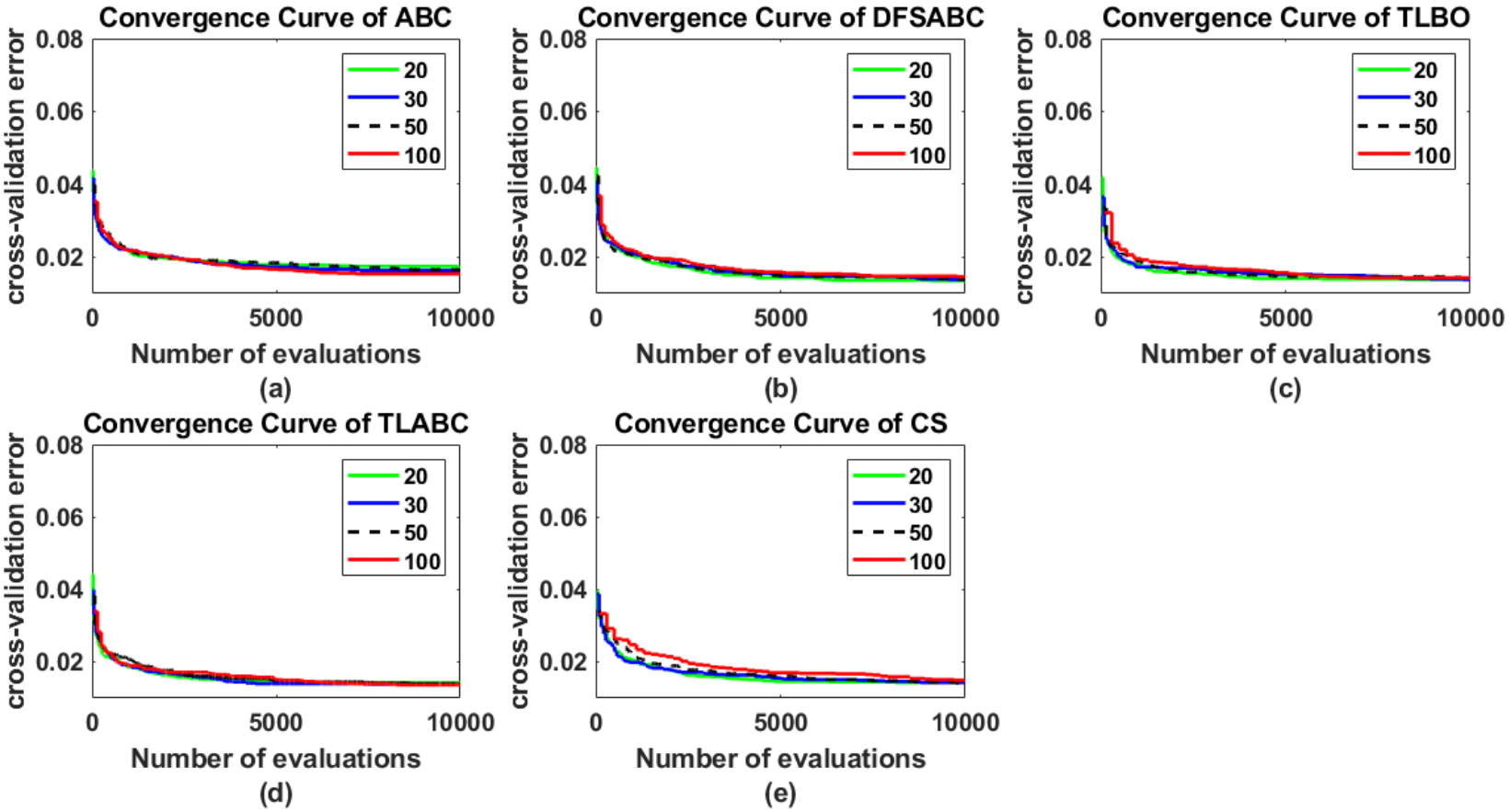

Figure 5.2 Convergence curves of meta-heuristic algorithms for Polynomial SVMs (first stage)
(a) ABC, (b) DFSABC, (c) TLBO, (d) TLABC, (e) CS

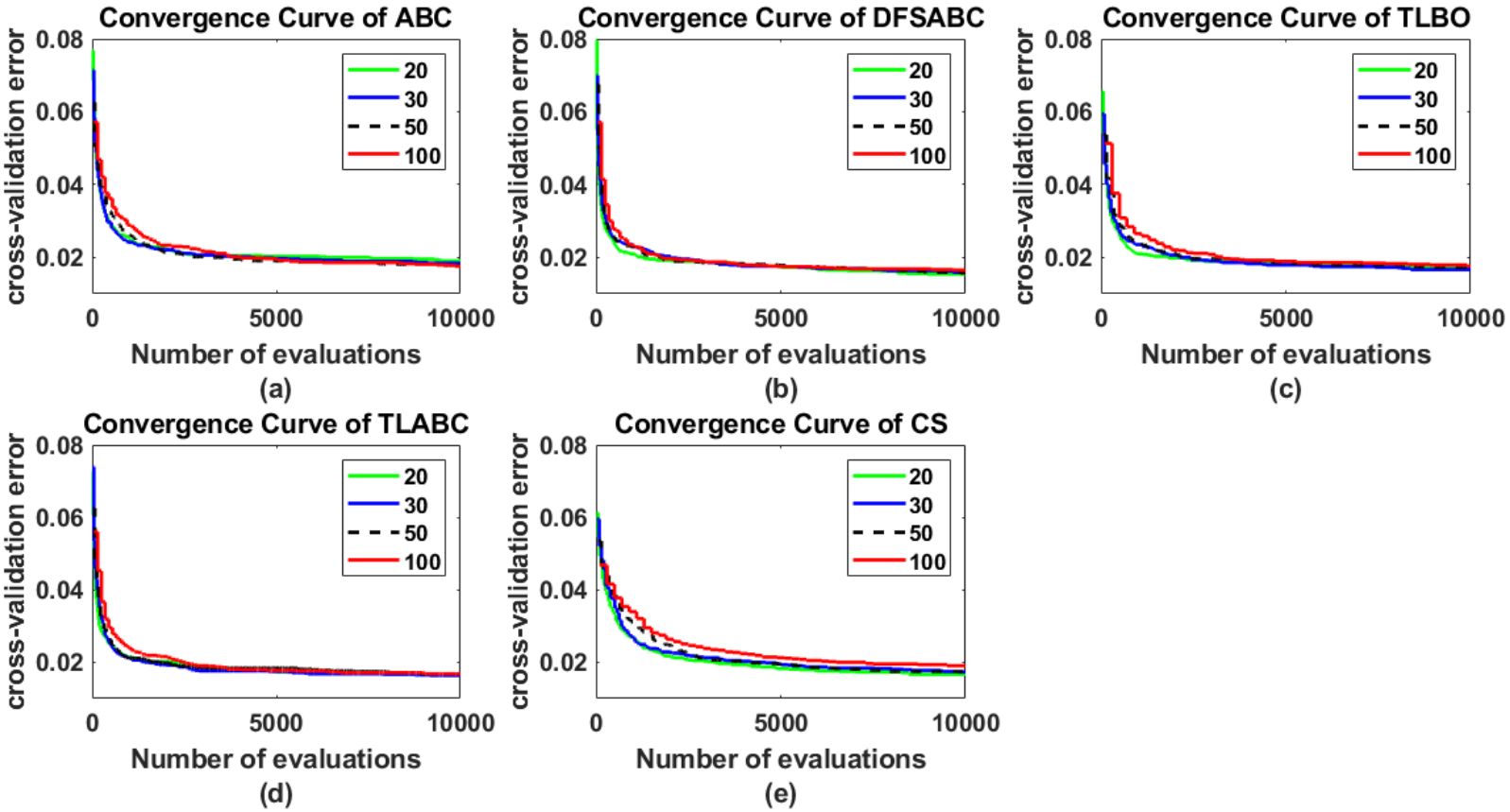

Figure 5.3 Convergence curves of meta-heuristic algorithms for Sigmoid SVMs (first stage)
(a) ABC, (b) DFSABC, (c) TLBO, (d) TLABC, (e) CS

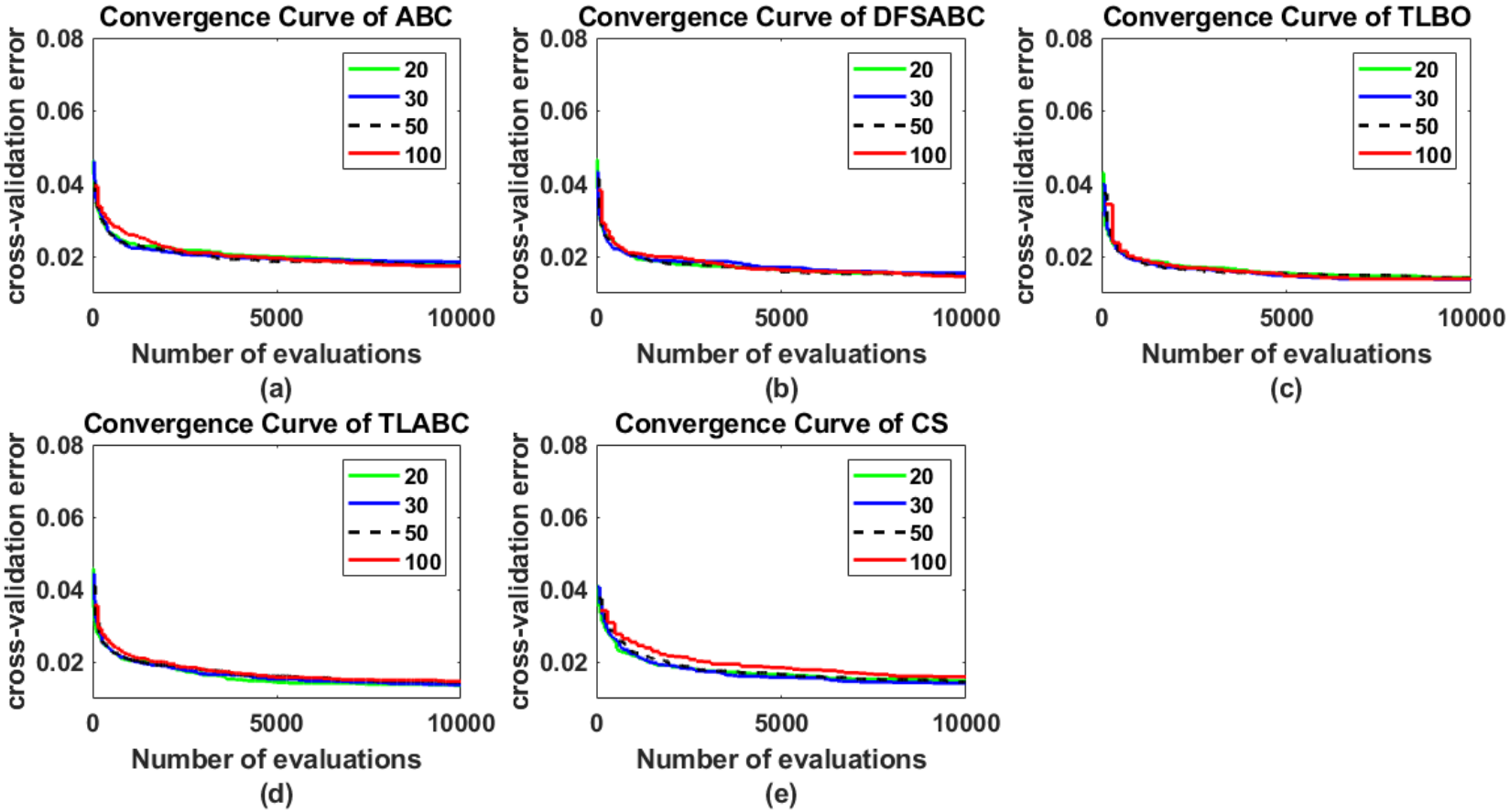

Figure 5.4 Convergence curves of meta-heuristic algorithms for hybrid-kernel SVMs (first stage)
(a) ABC, (b) DFSABC, (c) TLBO, (d) TLABC, (e) CS

5.2.1 Convergence Curves

First, Figure 5.5 shows the convergence curves of the five meta-heuristic algorithms when applied to Gaussian SVMs. Overall, TLABC and DFSABC exhibit the best convergence, while ABC possesses the worst convergence among the five algorithms. In addition, TLBO and CS show almost the same convergence

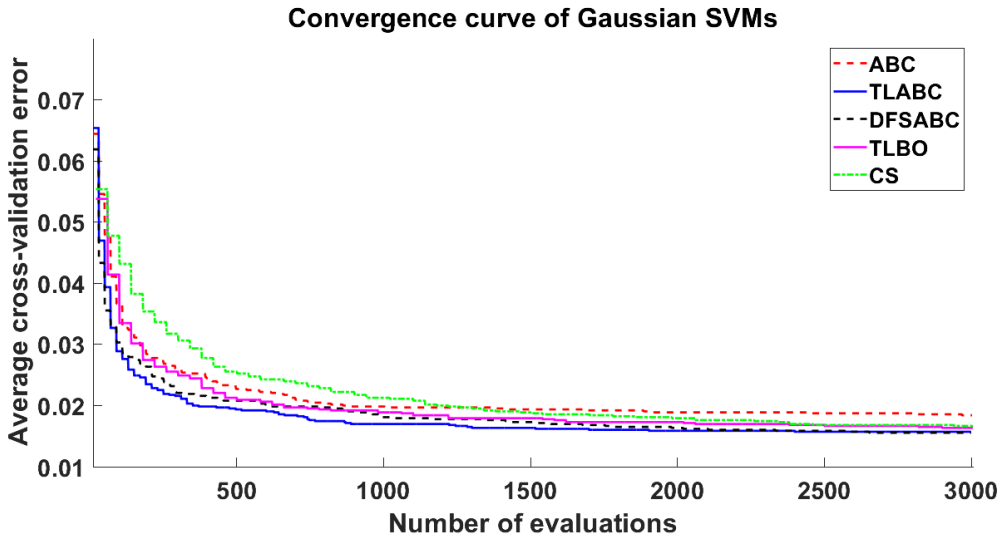

Figure 5.5 Convergence curves of meta-heuristic algorithms for Gaussian SVMs

Next, the five meta-heuristic algorithms are applied to polynomial SVMs and Figure 5.6 depicts their convergence curves. Generally, TLABC shows the best convergence and TLBO and CS exhibit comparable convergence. On the other hand, ABC shows the worst convergence.

Then, Figure 5.7 shows the convergence curves of the meta-heuristic algorithms when applied to sigmoid SVMs. DFSABC, TLABC and TLBO present roughly the same convergence. Although ABC and CS show slow convergence at the beginning, CS converges quickly near the end. Overall, DFSABC, TLABC, TLBO and CS have good convergence, whereas ABC has the worst convergence.

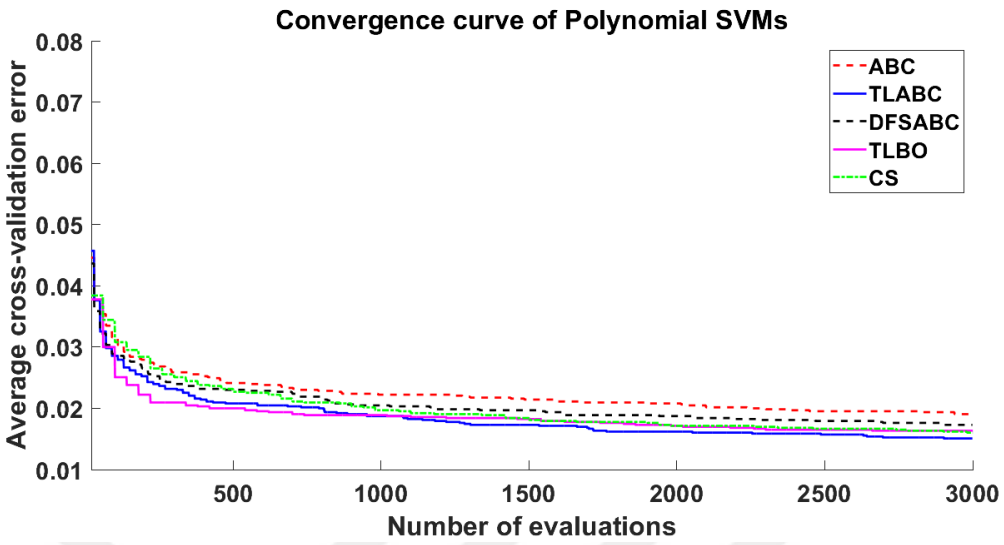

Figure 5.6 Convergence curves of meta-heuristic algorithms for polynomial SVMs

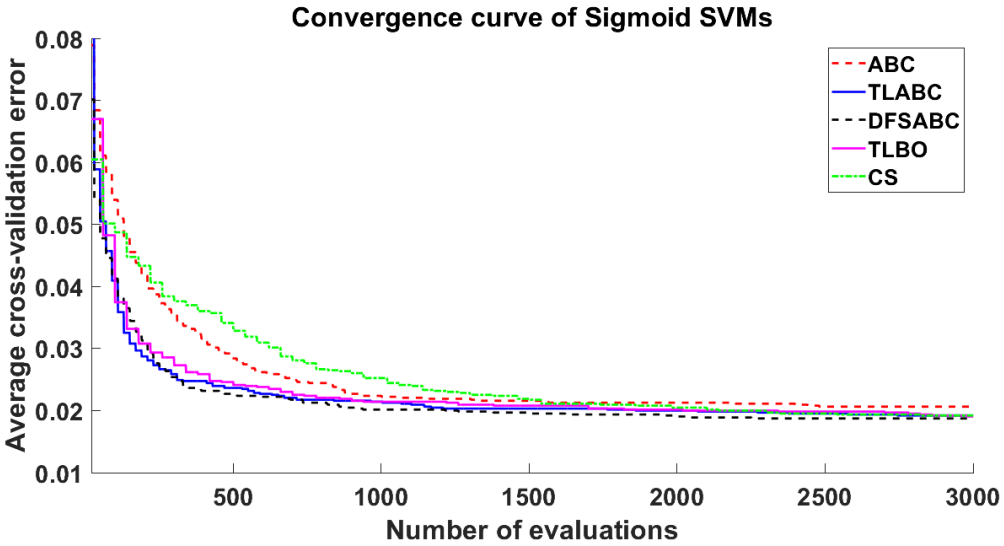

Figure 5.7 Convergence curves of the meta-heuristic algorithms for sigmoid SVMs

Finally, Figure 5.8 shows the convergence curves of the meta-heuristic algorithms when applied to hybrid-kernel SVMs. TLBO exhibits the best convergence. Also, DFSABC, TLABC and CS show nearly the same convergence. As always, ABC has the worst convergence.

To sum up, DFSABC, TLABC, TLBO and CS show good convergences, which are, in fact, similar to each other. In contrast, ABC always shows the worst convergence as compared with the other four algorithms.

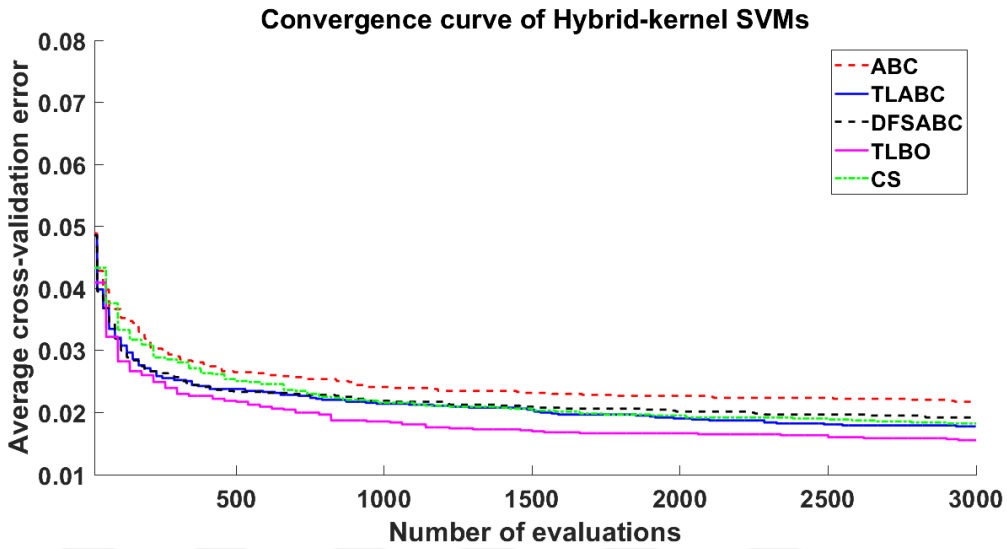

Figure 5.8 Convergence curves of meta-heuristic algorithms for hybrid-kernel SVMs

5.2.2 Average Classification Error

Here, the meta-heuristic-algorithm-optimized SVM classifiers are compared according to their average classification error of the 30 runs. Table 5.1 shows the results for each combination of meta-heuristic algorithms and SVM classifiers. The best result for each SVM classifier (i.e., the best result in each column) is typed in bold, whilst the best result for each meta-heuristic algorithm (i.e., the best result in each row) is asterisked, and the best result in the whole table is put inside a rectangular. For Gaussian SVMs, the best average classification error is 0.0156, which is corresponding to a classification accuracy of 98.44%. This classification accuracy is obtained by DFSABC, although TLABC, TLBO and CS give comparable results. For polynomial SVMs, the best average classification error is 0.0151, which is equivalent to a classification accuracy of 98.49%. This best accuracy is obtained by TLABC despite the fact that both TLBO and CS give comparable results. For sigmoid SVMs, the best average classification error is 0.0187, which is equivalent to a classification accuracy of 98.13%, and it is obtained by DFSABC.

TLBO, TLABC and CS give results near this value, as well. For hybrid-kernel SVMs, the best average classification error is 0.0156, which is corresponding to a classification accuracy of 98.44%, and this accuracy is obtained by TLBO.

Table 5.1 Average classification errors of the SVM classifiers

Meta-heuristic Algorithms	**Gaussian SVMs**	**Polynomial SVMs**	**Sigmoid SVMs**	**Hybrid-kernel SVMs**
ABC	0.0184*	0.0190	0.0205	0.0217
TLABC	0.0157	**0.0151***	0.0192	0.0178
DFSABC	**0.0156***	0.0171	**0.0187**	0.0190
TLBO	0.0162	0.0163	0.0190	**0.0156***
CS	0.0165	0.0160*	0.0192	0.0181

* refers in each row to the best result across that row

In summary, DFSABC yields the minimum average classification error in Gaussian SVMs and sigmoid SVMs while TLABC leads to the minimum average classification error in polynomial SVMs and TLBO leads to the minimum average classification error in hybrid-kernel SVMs. Given that, one is tempted to claim that DFSABC achieves the best results in general. Nonetheless, the results of DFSABC, TLABC, TLBO and CS are close to each other and there is a reasonable need to conduct statistical tests to prove, for example, whether DFSABC statistically outperforms the other in the cases of Gaussian and sigmoid SVMs. These statistical tests are presented in Section 5.3.

The above-drawn comparison between the meta-heuristic algorithms are conducted vertically with respect to Table 5.1. Now, if one needs to draw a comparison between the four SVM classifiers for each meta-heuristic algorithm, the comparison has to be conducted horizontally. For ABC and DFSABC, the minimum classification errors (i.e., 0.0184 and 0.0156 respectively) are obtained when using Gaussian SVMs. For TLABC and CS, the minimum classification errors (i.e., 0.0151 and 0.0160) are obtained when using polynomial SVMs. For TLBO, the minimum classification error (i.e., 0.0156) is achieved using hybrid-kernel SVMs.

To make the comparison much clearer, the rank of each combination of the SVM classifiers and meta-heuristic algorithms can be calculated. Table 5.2 shows the rank of each combination where the results of all the combinations are ordered and ranked ascendingly. Then, for each SVM classifier, the sum of ranks is calculated and the one with the lowest sum of rank is the best. It is obvious that Gaussian SVMs have the lowest sum (i.e., 32) and polynomial SVMs have the second lowest sum. Hence, Gaussian SVMs seem to rank first while polynomial SVMs seem to rank second according to Table 5.2.

Table 5.2 Ranks of SVM-algorithm-and-metaheuristic-algorithm combinations

SVM classifier	**Meta-heuristic Algorithm**	**Classification Error**	**Rank**	**Sum of Rank**
Gaussian SVMs	ABC	0.0184	12	**32**
	TLABC	0.0157	4	
	DFSABC	0.0156	2	
	TLBO	0.0162	6	
	CS	0.0165	8	
Polynomial SVMs	ABC	0.0190	14	36
	TLABC	**0.0151**	1	
	DFSABC	0.0171	9	
	TLBO	0.0163	7	
	CS	0.0160	5	
Sigmoid SVMs	ABC	0.0205	19	80
	TLABC	0.0192	17	
	DFSABC	0.0187	13	
	TLBO	0.0190	14	
	CS	0.0192	17	
Hybrid-kernel SVMs	ABC	0.0217	20	57
	TLABC	0.0178	10	
	DFSABC	0.0190	14	
	TLBO	0.0156	2	
	CS	0.0181	11	

To conclude, Gaussian SVMs seem to perform better than polynomial, sigmoid and hybrid-kernel SVMs. Like the above-raised argument, the results are close to each other and only statistical test can confirm whether there is a statistically significant difference between the results or not. In case of the existence of that difference, it can be argued that the Gaussian SVMs, indeed, outperform the other three.

5.2.3 Computational Time

Table 5.3 shows the average computational time for each SVM-algorithm-and-metaheuristic-algorithm combination where each combination is run 30 times and the average is reported. In Table 5.3, there exist two independent directions for comparisons: horizontal and vertical directions. When the results are compared vertically, one column at a time, the meta-heuristic algorithms are compared against each other for each SVM classifier. On the other hand, when the results are compared horizontally, one row at a time, the SVM classifiers are compared against each other for each meta-heuristic algorithm. The best result for each SVM classifiers (i.e., in each column) are typed in bold and the best result for each meta-heuristic algorithm (i.e., in each row) is marked by an asterisk. The time is measured in seconds. The platform used in this work is a Microsoft Azure virtual machine with Ubuntu 18.04.4 LTS (Linux) operating system, Intel® Xeon® Platinum 8168 CPU @ 2.70 GHz (only one of its cores with two threads) and 4 GB RAM. All the computations are done on MATLAB 2020a.

Table 5.3 Average computational time in seconds for meta-heuristic algorithms

Meta-heuristic Algorithms	**Gaussian SVMs**	**Polynomial SVMs**	**Sigmoid SVMs**	**Hybrid-kernel SVMs**
ABC	340.51*	**407.54**	423.32	571.77
TLABC	**321.71***	521.4	377.24	597.19
DFSABC	325.72*	475.42	370.41	**544.18**
TLBO	324.68*	747.07	**362.78**	848.35
CS	323.14*	697.46	372.4	699.1

For Gaussian SVMs, the best meta-heuristic algorithm in terms of time consumption is TLABC (around 5.4 minutes) in spite of the fact that the other four meta-heuristic algorithms show close results to this best result. In case of polynomial SVMs, ABC takes the shortest time (i.e., roughly 6.8 minutes) while TLBO takes the longest time (i.e., roughly 12.5 minutes). For sigmoid SVMs, TLBO is the fastest with the shortest time (i.e., just above 6 minutes) whereas the other four show slightly comparable results. For the hybrid-kernel SVMs, DFSABC takes the shortest time (i.e., just above 9 minutes) while TLBO takes the longest time (i.e., 14.1 minutes). As before, the figures of some of these meta-heuristic algorithms are close to each other and statistical tests are required to check if there is any statistically significant difference in the results.

Now, a comparison is conducted between the four SVM classifiers in terms of computational time for each meta-heuristic algorithm. For all the five meta-heuristic algorithms, Gaussian SVMs are the fastest with the shortest computational time. On the other hand, hybrid-kernel SVMs always take the longest time for all the meta-heuristic time. Also, it is noticeable that sigmoid SVMs show better results than polynomial SVMs.

In addition, Figure 5.9 shows the bar chart of the time consumption of each meta-heuristic algorithm and each SVMs classifier. For Gaussian and sigmoid SVMs, the results of the five meta-heuristic algorithms are approximately similar to each other. On the other hand, there are some discrepancies among the results of the five meta-heuristic algorithms in polynomial and hybrid-kernel SVMs. In general, Gaussian and sigmoid SVMs takes shorter time than polynomial and hybrid-kernel SVMs. Although there is a noticeable difference between the results of Gaussian and sigmoid SVMs and those of polynomial and hybrid-kernel SVMs, statistical tests can prove formally and mathematically that difference. The subsequent subsection provides details about the statistical analysis of the results.

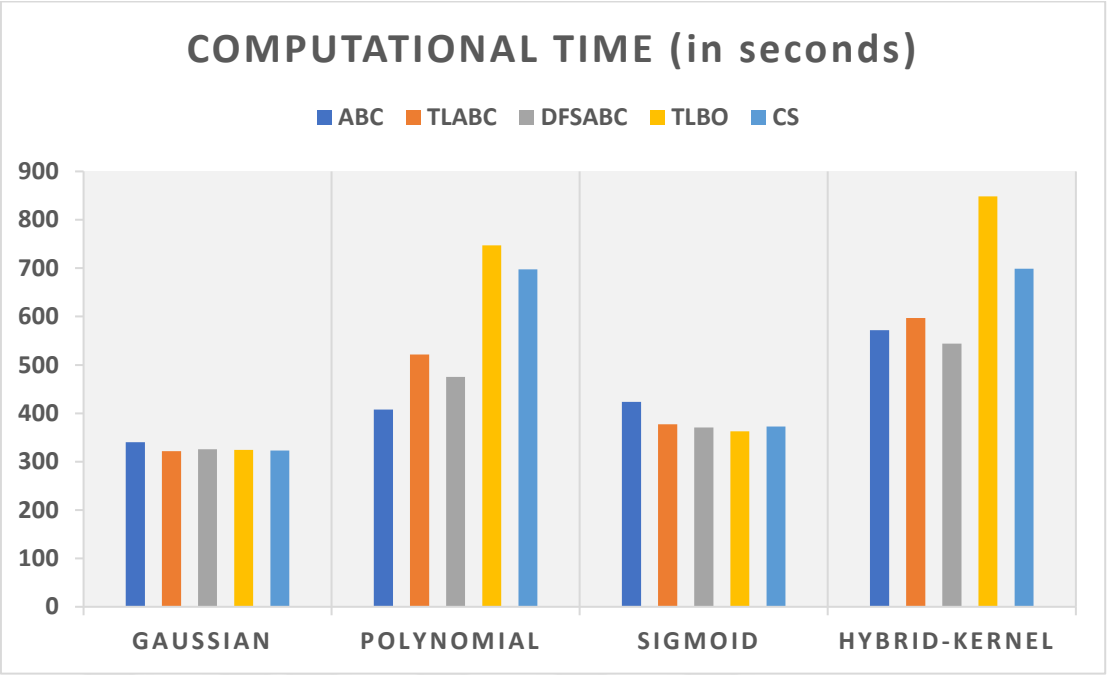

Figure 5.9 Bar chart of computational time

5.3 Overview of Statistical Analysis

Table 5.1 and Table 5.3 present the average classification error and average computational time of 30 runs. From these tables, the best meta-heuristic algorithm and the best SVM classifier can be determined according to their average figures. Nevertheless, some of these results look close to each other and the difference between them is small. Therefore, statistical tests can demonstrate whether there is actually a significant difference in the results or not. If the significant difference turns out to exist, the algorithm with the best average result is statistically proved to outperform the other algorithms. In these statistical tests, each algorithm is considered as a separate sample whose observations are the values of the performance criterion (i.e., either the classification error or the computational time) in the 30 runs. For instance, when analyzing statistically the results of the five meta-heuristic algorithms, there exist five samples, each of which has 30 observations corresponding to the 30 runs. Since, in this setting, there are more than two samples with unrelated data or observations, the Kruskal-Wallis test [90] has to be applied first to detect if all the samples (i.e., algorithms) are the same. Then, if this hypothesis is rejected indicating the existence of a potential difference between the samples, pairwise comparisons are conducted to compare each pair of

algorithms. The Wilcoxon rank sum test [91] is used in this work to draw these follow-up pairwise comparisons. Both tests are nonparametric tests where, unlike parametric tests, the data are no longer needed to assume to be normally distributed. In fact, the results of meta-heuristic algorithms are most likely to violate the assumptions of parametric tests, especially the normality assumption [92]. Therefore, nonparametric tests are more suitable when analyzing meta-heuristic algorithms.

The Kruskal-Wallis test compares the medians of three or more independent (i.e., unrelated) samples. The null hypothesis H_0 is stated as "all the samples are drawn from the same distribution", which means there is no significant difference between them, whilst the alternative hypothesis H_a is stated as "all the samples are not drawn from the same distribution", which indicates that at least two of them are different. For any statistical test, a level of significance α has to be specified. This level of significance is defined as the probability of the test to reject H_0 while it is true. Normally, α is specified as 0.1 or 0.05. As α decreases, it becomes more difficult to reject H_0 and the test becomes more conservative. In addition to α, another important quantity must be explained, which is known as the *p-value*. The p-value for given data is defined as the smallest level of significance α at which it is still possible to reject H_0. Accordingly, when testing any sample(s), the smaller the p-value is, the more confident we are to reject H_0. Also, if the p-value of some data is less than the significance level of that test, H_0 will be rejected in favor of H_a and we are at least 95% (if the significance level α is 0.05) confident that the alternative hypothesis H_a is true. Otherwise, we fail to reject the null hypothesis H_0. The failure of rejecting H_0 does not necessarily imply that H_0 is true. Instead, it means there is not enough evidence in the data to support the alternative hypothesis H_a.

Unlike the Kruskal-Wallis test, the Wilcoxon rank sum test compares the medians of only two independent samples, so it is a pairwise test. The null hypothesis H_0 is stated as "the two samples are drawn from the same distribution" whereas the alternative hypothesis H_a is stated as "the two samples are drawn from different distributions", which means there is a significant difference between these two samples.

It should be noted that the statistical tests have a binary decision, which means they either reject H_0 or fail to reject H_0 at a predefined significance level. The most widely used significance levels are 0.1, 0.05 and 0.01, which are equivalent to 90%, 95% and 99% confidence, respectively, that the null hypothesis H_0 is false. For example, if the

significance level of the test is 0.1 and the p-value of the data at hand is less than 0.1, then the test rejects H_0 with at least 90% confidence. This decision, however, does not reflect how much confident we are about rejecting H_0; instead, it gives only a lower bound on the confidence. On the other hand, the degree of confidence is embedded in the p-value. Hence, the p-values can be assigned into non-overlapping intervals such that more information about the degree of confidence are extracted from these p-values and expressed explicitly in the test decision. Doing so makes the test have more than two decisions, in fact, and then the results can be interpreted more accurately. Specifically, we let the statistical test have four decisions as follows:

- **Decision A:** "the test fails to reject H_0". This occurs whenever the p-value of the data is greater than 0.1.
- **Decision B:** "the test rejects H_0 with slight confidence". This occurs whenever the p-value of the data is less than 0.1 but greater than 0.05. Then, the test rejects H_0 with at least 90% confidence.
- **Decision C:** "the test rejects H_0 with sufficient confidence". This occurs whenever the p-value of the data is less than 0.05 but greater than 0.01. Then, the test rejects H_0 with at least 95% confidence.
- **Decision D:** "the test rejects H_0 with high confidence". This occurs whenever the p-value of the data is less than 0.01. Then, the test rejects H_0 with at least 99% confidence.

First, the results of the meta-heuristic algorithms will be compared statistically. Then, the results of the SVM classifiers will be compared statistically.

5.4 Statistical Analysis of Meta-heuristic Algorithms

In this subsection, the statistical analysis of the data related to both the classification error and computational time is presented. In fact, there are two steps: the first one is the Kruskal-Wallis test while the second one is the Wilcoxon rank sum test.

5.4.1 Kruskal-Wallis Test

First, the Kruskal-Wallis test is applied to the classification error data and computational time data of the five meta-heuristic algorithms where the meta-heuristic algorithms of every SVM classifier is compared separately. Table 5.4 shows the p-values and Chi-squared values for each SVM classifier. For simplicity, whenever the p-value is less than 0.0001, it is written in the table as "<1E-04".

Concerning the classification error, for the four SVM classifiers, their p-values are less than 0.05 (equivalently, their Chi-squared values are greater than the critical Chi-squared value equivalent to a significant level of 0.05 in a five-sample Kruskal-Wallis test, which is 9.49. Interested readers are referred to the Chi-square probabilities [93]. In fact, the Kruskal-Wallis test statistic is χ^2-distributed with (k-1) degrees of freedom where k is the number of samples). As a result, for each SVM classifier, H_0 is rejected and the five meta-heuristic algorithms are not the same when comparing their classification errors.

Regarding the computational time, for Gaussian SVMs, the p-value is 0.24 and then the test fails to reject the null hypothesis H_0 at either 95% significance level or even the more tolerant 90% significance level. Therefore, the test fails to prove any significant difference among the computational time data of the meta-heuristic algorithms and, consequently, all the algorithms seem to take the same time in Gaussian SVMs. For the other three SVM classifiers, their p-values are smaller than 0.0001; as a result, their computational time data are most likely to be significantly different with high confidence.

Table 5.4 Kruskal-Wallis test for meta-heuristic algorithms

SVM classifier	**Gaussian SVMs**	**Polynomial SVMs**	**Sigmoid SVMs**	**Hybrid-kernel SVMs**
Classification Error				
p-value	<1E-04	<1E-04	0.048	<1E-04
Chi-squared value	23.63	29.98	9.60	49.11
Computational Time				
p-value	0.24	<1E-04	<1E-04	<1E-04
Chi-squared value	5.47	90.93	104.69	90.63

3.4.2 Wilcoxon Rank Sum Test

Since it has just been proven that all the five meta-heuristic algorithms were not the same and they were not drawn from the same population, we can proceed to compare each pair of these five meta-heuristic algorithms by means of the Wilcoxon rank sum test.

Table 5.5 shows the p-values of this test for the Gaussian SVMs. According to these p-values, the classification error data of ABC are significantly different, with high confidence, from those of the other four meta-heuristic algorithms. However, the classification error data of these four meta-heuristic algorithms are not significantly different from each other since the test fails to reject the null hypothesis H_0. To conclude, for Gaussian SVMs, DFSABC has the best classification error. Although, we statistically prove that DFSABC is better than ABC since their data have significant difference, we fail to prove that DFSABC is better than TLABC, TLBO and CS due to the absence of significant difference between their results. Thus, DFSABC, TLBO, TLABC and CS show the same performance with Gaussian SVMs.

Concerning the computational time data, the Kruskal-Wallis test fails to reject the null hypothesis, which means that the computational time data of all the five meta-heuristic algorithms are the same with no significant difference. Thus, there is no point of conducting any follow-up tests to pairwisely compare the meta-heuristic algorithms.

Table 5.6 shows the p-values for polynomial SVMs. According to these p-values, we are highly confident that the classification error data of ABC are significantly different from those of the other four meta-heuristic algorithms. In addition, we are sufficiently confident that the classification error data of TLABC and DFSABC are significantly different, and slightly confident that those of TLABC and TLBO are significantly different. Nonetheless, the test fails to prove any significant difference between the remaining pairs. To conclude, for polynomial SVMs, TLABC has the best classification error. The test statistically proves with an acceptable significance level that TLABC is better than ABC, DFSABC and TLBO as there are significant differences among them. However, it concludes that TLABC and CS exhibit the same performance in polynomial SVMs.

Table 5.5 Wilcoxon rank sum test for meta-heuristic algorithms in case of Gaussian SVMs (Decision types are explained near the end of Section 5.3)

Meta-heuristic algorithm pairs	**Classification Error**		**Computational Time**	
	p-value	**Decision Type**	**p-value**	**Decision Type**
ABC vs **TLABC**	<1E-04	Decision D	-	-
ABC vs **DFSABC**	<1E-04	Decision D	-	-
ABC vs **TLBO**	0.0011	Decision D	-	-
ABC vs **CS**	0.0048	Decision D	-	-
TLABC vs **DFSABC**	0.7836	Decision A	-	-
TLABC vs **TLBO**	0.4259	Decision A	-	-
TLABC vs **CS**	0.1910	Decision A	-	-
DFSABC vs **TLBO**	0.2814	Decision A	-	-
DFSABC vs **CS**	0.1129	Decision A	-	-
TLBO vs **CS**	0.6114	Decision A	-	-

Regarding the computational time data, the computational time of ABC is believed with high confidence to be significantly different from TLBO, TLABC and CS. Yet, ABC and DFSABC are believed to take the same time since the test fails to reject that their data come from the same population (i.e., distribution). Similarly, it is concluded that there exists no significant difference between the computational time data of DFSABC and TLABC. However, the computational time data of TLBO are significantly different from TLABC and DFSABC with high confidence and significantly different from CS with slight confidence. It is also believed with high confidence that CS is significantly different from both TLABC and DFSABC. To conclude, for polynomial SVMs, ABC takes the shortest time and, according to the test, it is statistically significantly better than TLABC, TLBO and CS in terms of computational time. Nonetheless, the computational time data of ABC and DFSABC, which takes the second shortest time, are similar according to the test. Then, it is concluded that ABC outperforms TLABC, TLBO and CS in terms of computational time. Also, DFSABC is statistically proved to be faster than TLBO and CS in polynomial SVMs.

Table 5.6 Wilcoxon rank sum test for meta-heuristic algorithms in case of polynomial SVMs (Decision types are explained near the end of Section 5.3)

Meta-heuristic algorithm pairs	**Classification Error**		**Computational Time**	
	p-value	**Decision Type**	**p-value**	**Decision Type**
ABC vs **TLABC**	<1E-04	Decision D	<1E-04	Decision D
ABC vs **DFSABC**	0.0106	Decision C	0.2170	Decision A
ABC vs **TLBO**	0.0002	Decision D	<1E-04	Decision D
ABC vs **CS**	<1E-04	Decision D	<1E-04	Decision D
TLABC vs **DFSABC**	0.0210	Decision C	0.1580	Decision A
TLABC vs **TLBO**	0.0615	Decision B	<1E-04	Decision D
TLABC vs **CS**	0.1583	Decision A	<1E-04	Decision D
DFSABC vs **TLBO**	0.4970	Decision A	<1E-04	Decision D
DFSABC vs **CS**	0.2590	Decision A	<1E-04	Decision D
TLBO vs **CS**	0.6073	Decision A	0.0635	Decision B

Table 5.7 shows the p-values for sigmoid SVMs. According to these p-values, we are slightly confident that the classification error data of ABC and TLABC are significantly different, while we are sufficiently confident that ABC is different from both DFSABC and CS, and we are highly confident that ABC and TLBO are significantly different from one another. On the other hand, the test fails to reject the null hypotheses of the remaining pairs, which means that there is no significant difference among these pairs. To conclude, for sigmoid SVMs, DFSABC has the minimum classification error. It is statistically proved that DFSABC is better than ABC as there is a significant difference between them. Nevertheless, the statistical test concludes that TLBO, TLABC and CS have the same performance as DFSABC in the sigmoid SVMs.

As regards the computational time data, the computational time of ABC is believed with high confidence to be significantly different from all the other algorithms. Similarly, the computational time data of TLBO are believed with high confidence to be significantly different from all the other algorithms, as well. The computational time data of TLABC and DFSABC are significantly different with sufficient confidence. Finally, the computational time data of CS is not significantly different from either TLABC or DFSABC. To conclude, for sigmoid SVMs, TLBO is the one that takes the shortest time

and, according to the test, it is statistically significantly the best as its computational time data are significantly different from those of the others.

Table 5.7 Wilcoxon rank sum test for meta-heuristic algorithms in case of sigmoid SVMs (Decision types are explained near the end of Section 5.3)

Meta-heuristic algorithm pairs	**Classification Error**		**Computational Time**	
	p-value	**Decision Type**	**p-value**	**Decision Type**
ABC vs **TLABC**	0.0615	Decision B	<1E-04	Decision D
ABC vs **DFSABC**	0.0130	Decision C	<1E-04	Decision D
ABC vs **TLBO**	0.0096	Decision D	<1E-04	Decision D
ABC vs **CS**	0.0264	Decision C	<1E-04	Decision D
TLABC vs **DFSABC**	0.4992	Decision A	0.0436	Decision C
TLABC vs **TLBO**	0.7813	Decision A	<1E-04	Decision D
TLABC vs **CS**	0.9920	Decision A	0.1224	Decision A
DFSABC vs **TLBO**	0.5822	Decision A	<1E-04	Decision D
DFSABC vs **CS**	0.4410	Decision A	0.2282	Decision A
TLBO vs **CS**	0.7477	Decision A	<1E-04	Decision D

Table 5.8 shows the p-values for hybrid-kernel SVMs. According to these p-values, we are highly confident that the classification error data of ABC are significantly different from the other four. Likewise, we are highly confident that the classification error data of TLBO are significantly different from the other four. However, the test fails to prove any significant difference between the remaining pairs. To conclude, for hybrid-kernel SVMs, TLBO has the minimum classification error and the statistical test proves that it is definitely the best one as its classification error data are significantly different to the others.

According to the p-values of the computational time data, the computational time of ABC and TLABC is believed to come from the same population because there is no significant difference between them. As regards all the remaining pairs, they are believed with high confidence to be significantly different from each other. To conclude, for hybrid-kernel SVMs, DFSABC takes the shortest time and, according to the test, it is statistically

significantly the best as its computational time data are significantly different from those of the others.

Table 5.8 Wilcoxon rank sum test for meta-heuristic algorithms in case of hybrid-kernel SVMs (Decision types are explained near the end of Section 5.3)

Meta-heuristic algorithm pairs	**Classification Error**		**Computational Time**	
	p-value	**Decision Type**	**p-value**	**Decision Type**
ABC vs **TLABC**	<1E-04	Decision D	0.6735	Decision A
ABC vs **DFSABC**	0.0013	Decision D	0.0002	Decision D
ABC vs **TLBO**	<1E-04	Decision D	<1E-04	Decision D
ABC vs **CS**	<1E-04	Decision D	<1E-04	Decision D
TLABC vs **DFSABC**	0.1483	Decision A	0.0006	Decision D
TLABC vs **TLBO**	0.0072	Decision D	<1E-04	Decision D
TLABC vs **CS**	0.6219	Decision A	0.0005	Decision D
DFSABC vs **TLBO**	<1E-04	Decision D	<1E-04	Decision D
DFSABC vs **CS**	0.2746	Decision A	<1E-04	Decision D
TLBO vs **CS**	0.0008	Decision D	0.0001	Decision D

Bearing in mind the above statistical analysis, an overall comparison between the five meta-heuristic algorithms needs to be drawn. In terms of classification error, ABC always shows the worst performance among these five algorithms, while the remaining four algorithms show almost similar overall performances. However, DFSABC, generally, is the best one among these four algorithms in terms of computational time. Thus, DFSABC can be considered as the best one.

5.5 Statistical Analysis of SVM Classifiers

5.5.1 Kruskal-Wallis test

Table 5.9 shows the p-values as well as the Chi-squared values of SVM classifiers for each meta-heuristic algorithm. As shown in Table 5.9**,** all the p-values are less than 0.0001 so the null hypothesis H_0 is rejected with high confidence. In other words, the classification-error-related data and computational-time-related data of all the four SVM

classifiers are not the same and at least one pair of them are different from one another. The next step is conducting pairwise comparisons to determine which pairs are similar and which pairs are different.

Table 5.9 Kruskal-Wallis test for SVM classifiers

Meta-heuristic algorithm	**ABC**	**DFSABC**	**TLBO**	**TLABC**	**CS**
Classification Error					
p-value	<1E-04	<1E-04	<1E-04	<1E-04	<1E-04
Chi-squared value	25.95	24.65	29.37	34.43	27.84
Computational Time					
p-value	<1E-04	<1E-04	<1E-04	<1E-04	<1E-04
Chi-squared value	100.25	71.62	101.93	100.81	100.52

5.5.2 Wilcoxon Rank Sum test

For ABC, Table 5.10 shows the p-values of different SVM classifier pairs. According to these p-values, the classification error data of Gaussian SVMs are significantly different from those of both sigmoid and hybrid-kernel SVMs with high confidence; however, Gaussian and polynomial SVMs are not significantly different from each other since the test fails to prove any significant difference between their classification error data. In addition, polynomial SVMs are significantly different from hybrid-kernel SVMs with high confidence and from sigmoid SVMs with sufficient confidence. Finally, sigmoid and hybrid-kernel SVMs seem significantly different from one another with slight confidence. To conclude, ABC achieves the minimum classification error with Gaussian SVMs and then the results of Gaussian SVMs are statistically significantly better than those of both sigmoid and hybrid-kernel SVMs according to the test. However, Gaussian SVMs performs the same as polynomial SVMs, which achieve the second smallest classification error, because the test fails to detect any significant difference between them.

Regarding the computational time, all the four SVM classifiers are significantly different from each other. In fact, it is highly confident that there exists a significant difference between each pair of them. To conclude, Gaussian SVMs takes the shortest time for ABC.

Therefore, it is statistically proven that Gaussian SVMs are the best in terms of computational time.

Table 5.10 Wilcoxon rank sum test for SVM classifiers in case of ABC (Decision types are explained near the end of Section 5.3)

Meta-heuristic algorithm pairs	**Classification Error**		**Computational Time**	
	p-value	**Decision Type**	**p-value**	**Decision Type**
Gaussian vs **Polynomial**	0.3257	Decision A	<1E-04	Decision D
Gaussian vs **Sigmoid**	0.0017	Decision D	<1E-04	Decision D
Gaussian vs **Hybrid-kernel**	<1E-04	Decision D	<1E-04	Decision D
Polynomial vs **Sigmoid**	0.0289	Decision C	0.0029	Decision D
Polynomial vs **Hybrid-kernel**	0.0004	Decision D	<1E-04	Decision D
Sigmoid vs **Hybrid-kernel**	0.0606	Decision B	<1E-04	Decision D

For TLABC, Table 5.11 shows the p-values of different SVM classifier pairs. According to these p-values, the classification error data of Gaussian SVMs are significantly different from those of both sigmoid and hybrid-kernel SVMs with high confidence. Likewise, the classification error data of polynomial SVMs are significantly different from those of both sigmoid and hybrid-kernel SVMs with high confidence. Nonetheless, Gaussian and polynomial SVMs are not significantly different from one another. Finally, sigmoid and hybrid-kernel SVMs are significantly different from one another with slight confidence. To conclude, TLABC achieves the minimum classification error with polynomial SVMs. Then the results of polynomial SVMs are statistically significantly better than those of both sigmoid and hybrid-kernel SVMs according to the test. However, Gaussian SVMs perform the same as polynomial SVMs for TLABC as the test fails to detect any significant difference between them.

In terms of computational time, all the four SVM classifiers are significantly different from each other with high confidence. To conclude, Gaussian SVMs takes the shortest time for TLABC. Therefore, for TLABC, Gaussian SVMs are the best in terms of computational time.

Table 5.11 Wilcoxon rank sum test for SVM classifiers in case of TLABC (Decision types are explained near the end of Section 5.3)

Meta-heuristic algorithm pairs	**Classification Error**		**Computational Time**	
	p-value	**Decision Type**	**p-value**	**Decision Type**
Gaussian vs **Polynomial**	0.3500	Decision A	<1E-04	Decision D
Gaussian vs **Sigmoid**	<1E-04	Decision D	<1E-04	Decision D
Gaussian vs **Hybrid-kernel**	0.0054	Decision D	<1E-04	Decision D
Polynomial vs **Sigmoid**	<1E-04	Decision D	<1E-04	Decision D
Polynomial vs **Hybrid-kernel**	0.0008	Decision D	0.0004	Decision D
Sigmoid vs **Hybrid-kernel**	0.0768	Decision B	<1E-04	Decision D

For DFSABC, Table 5.12 shows the p-values of the different SVM classifier pairs. According to these p-values, the classification error data of Gaussian SVMs are significantly different from those of both sigmoid and hybrid-kernel SVMs with high confidence and significantly different from those of polynomial SVMs with slight significance. The classification error data of polynomial SVMs are significantly different from those of both sigmoid and hybrid-kernel SVMs with sufficient confidence. Finally, sigmoid and hybrid-kernel SVMs are not significantly different from one another. To conclude, DFSABC achieves the minimum classification error with Gaussian SVMs. Therefore, it is deduced that Gaussian SVMs statistically significantly outperform all the other SVM classifiers.

Regarding the computational time, Gaussian SVMs are significantly different from the other three with high confidence. Also, sigmoid and hybrid-kernel SVMs are significantly different from one another with high confidence whilst polynomial SVMs are significantly different from both sigmoid and hybrid SVMs with slight confidence. To conclude, Gaussian SVMs takes the shortest time for DFSABC, as well. Therefore, for DFSABC also, Gaussian SVMs are the best in terms of computational time.

Table 5.12 Wilcoxon rank sum test for SVM classifiers in case of DFSABC (Decision types are explained near the end of Section 5.3)

Meta-heuristic algorithm pairs	**Classification Error**		**Computational Time**	
	p-value	**Decision Type**	**p-value**	**Decision Type**
Gaussian vs **Polynomial**	0.06582	Decision B	<1E-04	Decision D
Gaussian vs **Sigmoid**	<1E-04	Decision D	<1E-04	Decision D
Gaussian vs **Hybrid-kernel**	<1E-04	Decision D	<1E-04	Decision D
Polynomial vs **Sigmoid**	0.03488	Decision C	0.0537	Decision B
Polynomial vs **Hybrid-kernel**	0.02098	Decision C	0.0679	Decision B
Sigmoid vs **Hybrid-kernel**	0.68499	Decision A	<1E-04	Decision D

For TLBO, Table 5.13 shows the p-values of SVM classifier pairs. According to these p-values, the classification error data of Gaussian SVMs are not significantly different from either polynomial or hybrid-kernel SVMs. However, the classification error data of Gaussian SVMs are significantly different from those of sigmoid with high confidence and so do hybrid-kernel SVMs. Similarly, polynomial and sigmoid SVMs are significantly different from one another with high confidence. To conclude, TLBO achieves the minimum classification error with hybrid-kernel SVMs. Hence, the results of hybrid-kernel SVMs are statistically significantly better than those of sigmoid SVMs according to the test. However, Gaussian, Polynomial and hybrid-kernel SVMs perform the same since the test fails to detect any significant difference between them.

Table 5.13 Wilcoxon rank sum test for SVM classifiers in case of TLBO (Decision types are explained near the end of Section 5.3)

Meta-heuristic algorithm pairs	**Classification Error**		**Computational Time**	
	p-value	**Decision Type**	**p-value**	**Decision Type**
Gaussian vs **Polynomial**	0.8018	Decision A	<1E-04	Decision D
Gaussian vs **Sigmoid**	<1E-04	Decision D	<1E-04	Decision D
Gaussian vs **Hybrid-kernel**	0.4127	Decision A	<1E-04	Decision D
Polynomial vs **Sigmoid**	<1E-04	Decision D	<1E-04	Decision D
Polynomial vs **Hybrid-kernel**	0.2923	Decision A	0.0144	Decision C
Sigmoid vs **Hybrid-kernel**	<1E-04	Decision D	<1E-04	Decision D

Concerning the computational time, all the SVM classifiers are significantly different from each other at least with sufficient confidence according to the p-values in Table 5.13. To conclude, since Gaussian SVMs takes the shortest time for TLBO, Gaussian SVMs are the best in terms of computational time.

For CS, Table 5.14 shows the p-values of SVM classifier pairs. According to these p-values, the classification error data of Gaussian SVMs are significantly different from sigmoid SVMs with high confidence and from hybrid-kernel SVMs with sufficient confidence. However, the classification error data of Gaussian and polynomial SVMs are the same as there exists no significant difference between their data. Also, polynomial SVMs are significantly different from both sigmoid and hybrid-kernel SVMs with high confidence. Lastly, sigmoid and hybrid-kernel SVMs are not significantly different from each other. To conclude, CS achieves the minimum classification error with polynomial SVMs. Hence, the results of polynomial SVMs are statistically significantly better than those of sigmoid and hybrid-kernel SVMs according to the test. Still, Gaussian SVMs perform the same as polynomial SVMs since the test fails to detect any significant difference between them.

Table 5.14 Wilcoxon rank sum test for SVM classifiers in case of CS (Decision types are explained near the end of Section 5.3)

Meta-heuristic algorithm pairs	**Classification Error**		**Computational Time**	
	p-value	**Decision Type**	**p-value**	**Decision Type**
Gaussian vs **Polynomial**	0.4411	Decision A	<1E-04	Decision D
Gaussian vs **Sigmoid**	<1E-04	Decision D	<1E-04	Decision D
Gaussian vs **Hybrid-kernel**	0.0166	Decision C	<1E-04	Decision D
Polynomial vs **Sigmoid**	<1E-04	Decision D	<1E-04	Decision D
Polynomial vs **Hybrid-kernel**	0.0024	Decision D	0.5106	Decision A
Sigmoid vs **Hybrid-kernel**	0.1010	Decision A	<1E-04	Decision D

As regards the computational time, polynomial and hybrid-kernel SVMs take almost the same amount of computational time as the test fails to discover any significant difference in their data. Apart from that pair, all the other pairs are significantly different from one

another with high confidence. To conclude, since Gaussian SVMs takes the shortest time for CS as well, Gaussian SVMs are still the best in terms of computational time.

Given the above statistical analysis, an overall comparison between the four SVM classifiers needs to be drawn. In terms of classification error, Gaussian and polynomial SVMs obtain similar results and they are better than sigmoid and hybrid-kernel SVMs. Nonetheless, Gaussian SVMs are always the least computationally expensive. Moreover, sigmoid SVMs are better than polynomial and hybrid-kernel SVMs in terms of computational time. Finally, hybrid-kernel SVMs have the worst overall performance.

5.6 Comparison between Current Results and Literature

In this section, one of the best solutions found by DFSABC is used as the optimal SVM hyper-parameters. SVMs are retrained many enough times and the predicted class labels of the wheat varieties data are reported, along with their confusion matrix. Cross-validation is used in this work and it partitions the data randomly into multiple groups. This random partition may lead into different results every time. Therefore, SVMs will be retrained 100 times and both the best and average cross-validation classification accuracies will be stated. In addition, the confusion matrix of the best model with the best cross-validation classification accuracy will be shown.

Table 5.15 summarizes the classification accuracies of the four SVM classifiers, together with the hyper-parameters yielding those accuracies. The classification accuracies shown in the table are the average and the maximum (written as the best in the table) of 100 runs. Further, the standard deviations of these 100 runs are shown in the last column. For all the four SVMs, the tradeoff parameter C is relatively large so as to penalize adequately the training errors. This makes sure that the SVM model does not underfit the data. In Table 5.15, two points deserve consideration. The first one is that the polynomial degree for both polynomial and hybrid-kernel SVMs is one, which means that the polynomial kernel is linear. The other one is that the weight parameter of hybrid-kernel is very small (i.e., 0.053). That means a large weight is given to the polynomial part while the Gaussian part is given very small weight.

Table 5.15 SVM hyper-parameters and their classification accuracy

SVM Classifier	**Hyper-parameters**		**Classification Accuracy**		
	Name	**Value**	**Average**	**Best**	**Std. dev.**
Gaussian	Kernel scale (σ) Tradeoff parameter (C)	4.2 7.6×10^3	**96.06 %**	**98.57 %**	**0.96**
Polynomial	Polynomial degree (r) Tradeoff parameter (C)	1 3.76×10^3	95.67 %	98.10 %	1.06
Sigmoid	Kernel scale (γ) Kernel offset (δ) Tradeoff parameter (C)	0.13 1.43 1.34×10^4	95.52 %	98.10 %	1.04
Hybrid-kernel	Kernel scale (σ) Polynomial degree (r) Weight (β) Tradeoff parameter (C)	88.4 1 0.053 8.17×10^3	94.45 %	97.62 %	1.07

5.4.1 Confusion Matrix

Figure 5.10 shows the confusion matrix of the best Gaussian SVM model trained using the hyper-parameters shown in Table 5.15. The confusion matrix is a 3-by-3 matrix since we have three classes. Its rows (i.e., the y-axis) represent the true classes whilst its columns (i.e., the x-axis) represent the predicted classes which the best Gaussian SVM model outputs. Indeed, a great deal of useful information can be extracted from the confusion matrix. The diagonal elements of this matrix represent the correctly classified observations and the off-diagonal elements represent the misclassified observations. It also displays the class-wise recall (i.e., the class-wise true positive rate or also known as sensitivity) on the right as well as the class-wise precision (also known as confidence) on the bottom. The class-wise recall is the percentage of the class observations that are correctly classified into that class, while the class-wise precision is the percentage of the observations predicted to belong to that class actually belong to that class. Then the overall recall is the average of the recalls of the three classes and, likewise, the overall precision is the average of the precisions of the three classes. According to Figure 5.10,

in total, only three observations out of 210 observations are misclassified and, therefore, the classification accuracy is 98.57 %. Also, all the Rosa data are correctly classified while 98.6% of the Canadian data are correctly classified and 97.1% of the Kama data are correctly classified. Accordingly, the Rosa recall, Canadian recall and Kama recall are 100%, 98.6% and 97.1% respectively.

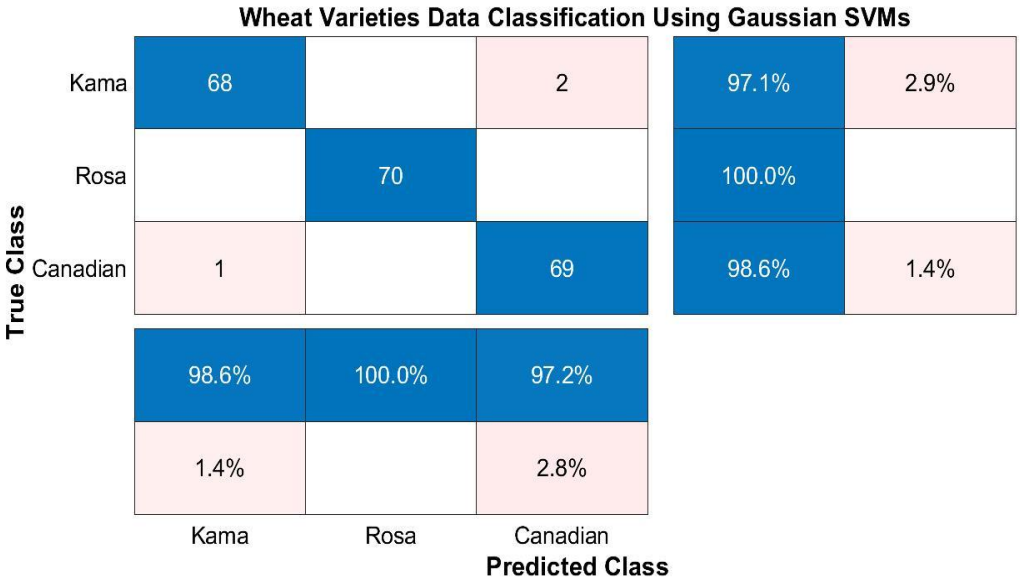

Figure 5.10 Confusion matrix of Gaussian SVMs

Figure 5.11 shows the confusion matrix of the best polynomial SVM model trained using the hyper-parameters shown in Table 5.15. Just four observations out of the 210 observations are misclassified; therefore, the total accuracy is 98.1%. Regarding the class-wise accuracy, 97.1 % of the Kama data are correctly classified where only two observations out of 70 Kama observations are misclassified as Canadian. Likewise, 97.1 % of Canadian observations are correctly classified where two observations out of its 70 observations are misclassified as Kama. In contrast, all the Rosa observations are correctly classified.

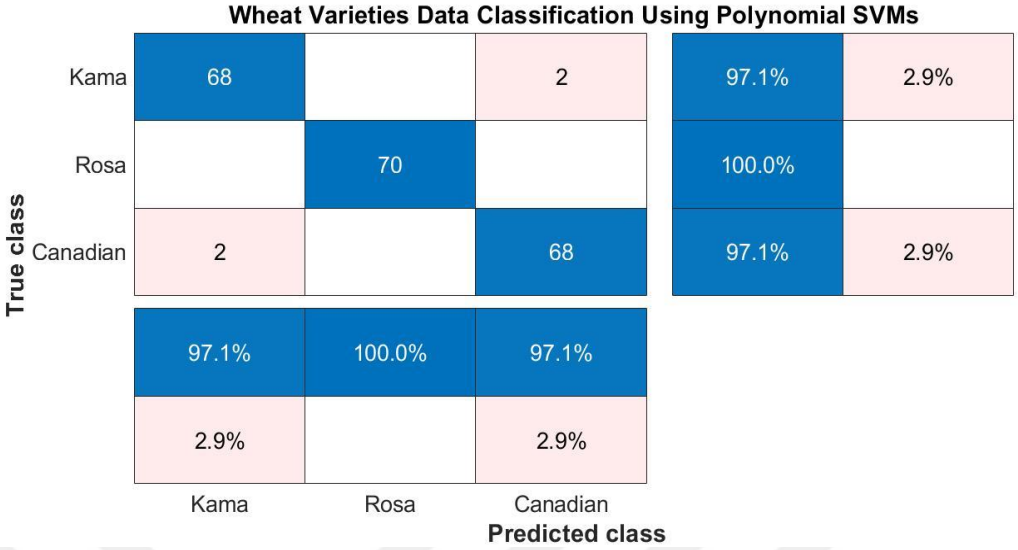

Figure 5.11 Confusion matrix of polynomial SVMs

Figure 5.12 shows the confusion matrix of the best sigmoid SVM model trained using the hyper-parameters shown in Table 5.15. Like polynomial SVMs, only four observations out of the 210 observations are misclassified. As regards the class-wise accuracy, all the 70 Rosa observations are correctly classified. However, 95.7% of the Kama observations are classified correctly where three of its observations are misclassified as Canadian.

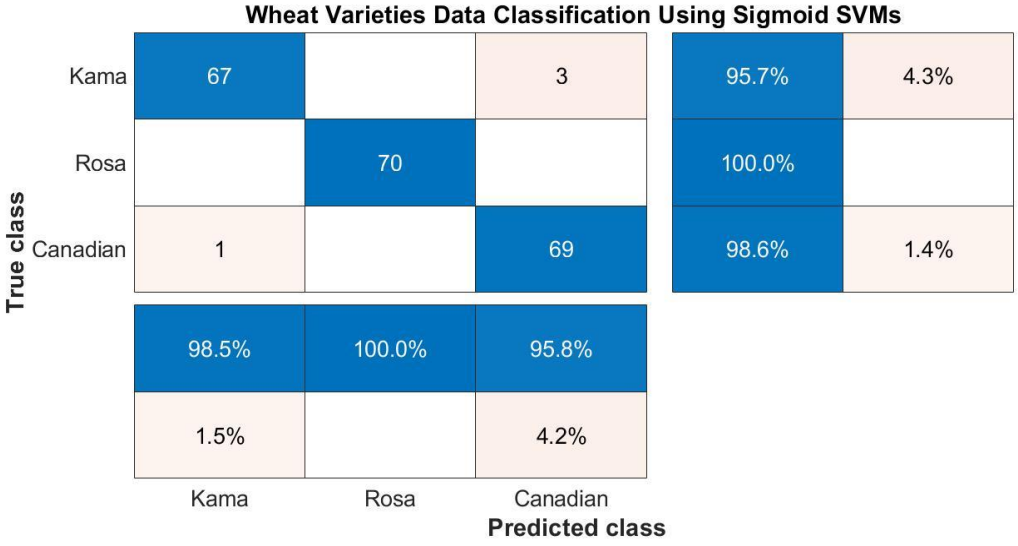

Figure 5.12 Confusion matrix of sigmoid SVMs

Similarly, 98.6% of the Canadian observations are classified correctly such that only one of its observation is misclassified as Kama.

Figure 5.13 shows the confusion matrix of the best hybrid-kernel SVM model trained using the hyper-parameters shown in Table 5.15. Only five observations out of the 210 observations are misclassified so the total classification accuracy is 97.62 %. Regarding the class-wise accuracy, all the Rosa observations are correctly classified. Nonetheless, 98.6% of the Canadian observations are correctly classified whereas 94.3% of the Kama observations are correctly classified. In fact, one Canadian observation is misclassified as Kama while two Kama observations are misclassified as Rosa and two other Kama observations are misclassified as Canadian.

To sum up, Gaussian SVMs is the best in terms of both the total classification accuracy and class-wise classification accuracies. Comparing the three wheat varieties, the Kama variety is always the most difficult one to classify while the Rosa variety is the easiest one to classify where all its observations are always correctly classified.

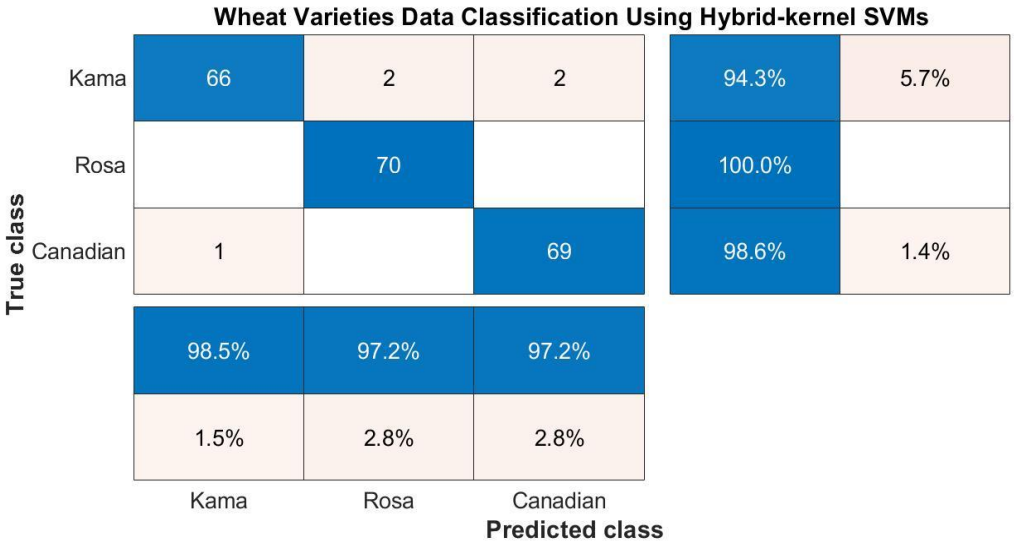

Figure 5.13 Confusion matrix of hybrid-kernel SVMs

5.4.2 Current Study vs Literature

First, Table 5.16 conducts a comparison between the current results of the four SVMs classifiers and the results of the complete gradient clustering and K-means clustering algorithms reported in the original paper of this dataset [10]. As shown in this table, all the four SVM classifiers lead to class-wise and overall classification accuracies better than those of [10].

Table 5.16 Comparison between the current study and the dataset original study [10]

Algorithm		**Kama**	**Rosa**	**Canadian**	**Total**
Clustering[10]	Complete Gradient	84 %	96 %	96 %	92 %
	K-Means	93 %	94 %	89 %	92 %
SVMs	Gaussian	**97.1** %	**100** %	**98.6** %	**98.57** %
(current results)	Polynomial	**97.1** %	**100** %	97.1 %	98.10 %
	Sigmoid	95.7 %	**100** %	**98.6** %	98.10 %
	Hybrid-kernel	94.3 %	**100** %	**98.6** %	97.62 %

Nevertheless, only two of them used the cross-validation scheme while the others used the holdout testing scheme. As previously raised, the holdout testing scheme is not suitable for this dataset due to its small size and, therefore, the cross-validation scheme is used in this work. Then, the current results are compared only against the two cross-validation-scheme-using studies. It is, indeed, pointless to compare them against the results of the holdout testing scheme since the holdout testing scheme is most likely to yield biased results with this small dataset. Moreover, the two previous studies shown in Table 5.17 used the ten-fold cross-validation and the current study uses the five-fold cross-validation, and it is raised in Section 4.5 that the latter is preferable to and more frequent than the former when the dataset is small. That is to ensure the testing fold has sufficient testing data for each model and the results are not biased, then. According to Table 2.17, all the four metaheuristic-algorithm-optimized SVM classifiers used in this study outperform the algorithms used in those two previous studies. Gaussian SVMs lead to the best results, though.

Table 5.17 Comparison between the current results and literature

Study	**Learning Algorithms**	**Classification Accuracy**
K. Sabancı, et al. [36]	MLP	97.1%
	KNN	95.7%
	J48	91.9%
	Naïve Bayes	91.4%
P. A. Kowalski et al. [38]	PNN with LSA	93.4%
Current Study	Gaussian	**98.57** %
(SVMs)	Polynomial	98.10 %
	Sigmoid	98.10 %
	Hybrid-kernel	97.62 %

CHAPTER 6
CONCLUSION

6.1 Summary

SVMs are indeed a powerful learning algorithm with high generalization capability. They are a sophisticated technique that can be used for classifying even complicated datasets. However, they involve some hyper-parameters playing a vital role in the overall performance of SVMs. The tradeoff parameter, which acts as a regularization parameter, and the kernel parameters constitute the SVM hyper-parameters. These hyper-parameters must be tuned according to the dataset at hand. In fact, efficiently tuning them improve the performance substantially. Accordingly, this study tunes the hyper-parameters of four different SVM algorithms; namely Gaussian, polynomial, sigmoid and hybrid-kernel SVMs (which are a weighted linear combination of Gaussian and polynomial SVMs); according to the wheat varieties dataset using five meta-heuristic optimization algorithms: ABC, DFSABC, TLBO, TLABC and CS, all of which are swarm intelligence algorithms. The data are first preprocessed using the min-max normalization prior to training and testing. Since this is a multiclass classification problem, SVMs are used with the one-versus-one scheme. The five-fold cross-validation error is used in this study as the objective function of the meta-heuristic algorithms and the performance measure of the SVM classifiers. It is, in fact, a reasonable unbiased estimate of the generalization error as the size of the dataset is small. The results of the four different SVM classifiers are statistically compared against each other in terms of the classification accuracy and computational time. Likewise, the five meta-heuristic algorithms are statistically compared against each other.

Regarding the meta-heuristic algorithms, ABC always shows the worst convergence with the least performance. It is also statistically proved that the other four meta-heuristic algorithms outperform ABC in this study. It is expected that DFSABC and TLABC perform better than ABC as they are modified, improved versions of ABC. On the other

hand, TLBO and CS show similar convergences to DFSABC and TLABC and the classification-accuracy-related results of all these four algorithms are roughly similar to each other. However, when the meta-heuristic algorithms are compared in terms of both their convergence and computational time, DFSABC yields the best results in most of the experiments and, therefore, it is considered as the best in general. Overall, TLABC can rank second.

Regarding the four SVM classifiers, Gaussian and polynomial SVMs have almost the same performance in terms of classification accuracy and both are better than sigmoid and hybrid-kernel SVMs. In terms of computational time, Gaussian SVMs is the best although there exists no significantly big difference between their computational-time-related results and those of sigmoid SVMs, which rank second in terms of computational time. Nonetheless, sigmoid SVMs are less computationally expensive than polynomial and hybrid-kernel SVMs. Overall, Gaussian SVMs rank first and sigmoid SVMs rank second since sigmoid SVMs less computationally expensive than polynomial SVMs and the difference between their classification-accuracy-related results is relatively small. So, in this dataset, sigmoid SVMs are preferable to both polynomial and hybrid-kernel SVMs. Also, hybrid-kernel SVMs are the worst among the four SVM classifiers.

Finally, the SVM classifiers are retrained by using one of the best solutions found by DFSABC as their hyper-parameters. The best classification accuracy of 98.57 % is achieved by Gaussian SVMs. In contrast, polynomial, sigmoid and kernel-hybrid SVMs achieve classification accuracies of 98.10%, 98.10% and 97.62%, respectively. All these figures are much better than the results reported in the original paper of the dataset. In addition, these current results are better than the previous studies which use the cross-validation scheme, just as does this study. Concerning the class-wise classification accuracy (also known as the class-wise recall), the class-wise classification accuracies of Gaussian SVMs are 97.1% for Kama, 100% for Rosa and 98.6% for Canadian. Indeed, Gaussian SVMs lead to the best class-wise classification accuracies, as well.

6.2 Future Study

In this section, some ideas and points are recommended as future work. There are, in fact, three recommendations:

1. The dataset used in this work is small where it contains only 210 observations. The proposed framework and methodology of this study can be applied to larger datasets, for example, with thousands of observations. More importantly, if the dataset is large enough, the holdout testing scheme can be used instead of the cross-validation scheme because the computational complexity of the former is less than the latter. In the case of a large dataset, the holdout testing error becomes a good unbiased estimate of the generalization error.
2. Five meta-heuristic optimization algorithms are used in this work to tune the SVM hyper-parameters. In fact, too many meta-heuristic optimization algorithms have been already proposed in the literature. Some of these algorithms can also be used for tuning the SVM hyper-parameters with different datasets. Then their results are compared with respect to their convergence and time consumption.
3. Finally, the optimization procedures proposed in this work using meta-heuristic algorithms can be applied to tune the hyper-parameters of learning algorithms other than SVMs.

REFERENCES

1. Abe, S., 2010. Support Vector Machines For Pattern Classification, Second Edition. Springer-Verlag London, , 473 pp.

2. Byun, H., Lee, S.W., 2002. Applications of support vector machines for pattern recognition: A survey, pp. 213–236, *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Springer Verlag, .

3. Karaboga, D., 2005. An idea based on Honey Bee Swarm for Numerical Optimization, pp. 10, *Technical Report TR06, Erciyes University*, Erciyes university, engineering faculty, computer engineering department, .

4. Karaboga, D., Basturk, B., 2008. On the performance of artificial bee colony (ABC) algorithm. **Applied Soft Computing Journal**, **8**(1): 687–697.

5. Chen, X., Xu, B., Mei, C., Ding, Y., Li, K., 2018. Teaching–learning–based artificial bee colony for solar photovoltaic parameter estimation. **Applied Energy**, **212**: 1578–1588.

6. Cui, L., Li, G., Lin, Q., Du, Z., Gao, W., Chen, J., Lu, N., 2016. A novel artificial bee colony algorithm with depth-first search framework and elite-guided search equation. **Information Sciences**, **367**–**368**: 1012–1044.

7. Rao, R. V., Savsani, V.J., Vakharia, D.P., 2011. Teaching-learning-based optimization: A novel method for constrained mechanical design optimization problems. **CAD Computer Aided Design**, **43**(3): 303–315.

8. Yang, X.S., Deb, S., 2009. Cuckoo search via Lévy flights, pp. 210–214, *2009 World Congress on Nature and Biologically Inspired Computing, NABIC 2009 - Proceedings*, .

9. Gandomi, A.H., Yang, X.S., Alavi, A.H., 2013. Cuckoo search algorithm: A metaheuristic approach to solve structural optimization problems. **Engineering with Computers**, **29**(1): 17–35.

10. Charytanowicz, M., Niewczas, J., Kulczycki, P., Kowalski, P.A., Łukasik, S., Zak,

S., 2010. Complete gradient clustering algorithm for features analysis of X-ray images. **Advances in Intelligent and Soft Computing**, **69**: 15–24.

11. Gorgevik, D., Cakmakov, D., Radevski, V., 2001. Handwritten digit recognition by combining support vector machines using rule-based reasoning, pp. 139–144, *Proceedings of the International Conference on Information Technology Interfaces, ITI*, University of Zagreb, .

12. Bahlmann, C., Haasdonk, B., Burkhardt, H., 2002. Online handwriting recognition with support vector machines - A kernel approach, pp. 49–54, *Proceedings - International Workshop on Frontiers in Handwriting Recognition, IWFHR*, .

13. Ahmad, A.R., Khalid, M., Viard-Gaudin, C., Poisson, E., 2004. Online handwriting recognition using support vector machine, *IEEE Region 10 Annual International Conference, Proceedings/TENCON*, Institute of Electrical and Electronics Engineers Inc., .

14. Guo, G., Li, S.Z., Chan, K.L., 2001. Support vector machines for face recognition. **Image and Vision Computing**, **19**(9–10): 631–638.

15. Jonsson, K., Kittler, J., Li, Y.P., Matas, J., 2002. Support vector machines for face authentication. **Image and Vision Computing**, **20**(5–6): 369–375.

16. Gumus, E., Kilic, N., Sertbas, A., Ucan, O.N., 2010. Evaluation of face recognition techniques using PCA, wavelets and SVM. **Expert Systems with Applications**, **37**(9): 6404–6408.

17. Ghimire, D., Lee, J., 2013. Geometric Feature-Based Facial Expression Recognition in Image Sequences Using Multi-Class AdaBoost and Support Vector Machines. **Sensors**, **13**(6): 7714–7734.

18. Ma, C., Randolph, M.A., Drish, J., 2001. A support vector machines-based rejection technique for speech recognition, pp. 381–384, *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, .

19. Zhang, S.X., Liu, C., Yao, K., Gong, Y., 2015. Deep neural support vector machines for speech recognition, pp. 4275–4279, *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, Institute

of Electrical and Electronics Engineers Inc., .

20. Kim, K.I., Jung, K., Park, S.H., Kim, H.J., 2000. Support vector machine-based text detection in digital video. **Neural Networks for Signal Processing - Proceedings of the IEEE Workshop**, **2**: 634–641.

21. Joachims, T., 1998. Text categorization with Support Vector Machines: Learning with many relevant features, pp. 137–142, *European Conference on Machine Learning*, Springer, Berlin, Heidelberg.

22. Anguita, D., Ghio, A., Oneto, L., Parra, X., Reyes-Ortiz, J.L., 2012. Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine, pp. 216–223, *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Springer, Berlin, Heidelberg, .

23. Tran, D.N., Phan, D.D., 2016. Human Activities Recognition in Android Smartphone Using Support Vector Machine, pp. 64–68, *Proceedings - International Conference on Intelligent Systems, Modelling and Simulation, ISMS*, IEEE Computer Society, .

24. Rumpf, T., Mahlein, A.K., Steiner, U., Oerke, E.C., Dehne, H.W., Plümer, L., 2010. Early detection and classification of plant diseases with Support Vector Machines based on hyperspectral reflectance. **Computers and Electronics in Agriculture**, **74**(1): 91–99.

25. Yao, Q., Guan, Z., Zhou, Y., Tang, J., Hu, Y., Yang, B., 2009. Application of support vector machine for detecting rice diseases using shape and color texture features, pp. 79–83, *2009 International Conference on Engineering Computation, ICEC 2009*, .

26. Islam, M., Dinh, A., Wahid, K., Bhowmik, P., 2017. Detection of potato diseases using image segmentation and multiclass support vector machine, *Canadian Conference on Electrical and Computer Engineering*, Institute of Electrical and Electronics Engineers Inc., .

27. Tellaeche, A., Pajares, G., Burgos-Artizzu, X.P., Ribeiro, A., 2011. A computer vision approach for weeds identification through Support Vector Machines.

Applied Soft Computing Journal, **11**(1): 908–915.

28. Kovačević, M., Bajat, B., Gajić, B., 2010. Soil type classification and estimation of soil properties using support vector machines. **Geoderma**, **154**(3–4): 340–347.

29. Eliopoulos, P.A., Potamitis, I., Kontodimas, D.C., 2016. Estimation of population density of stored grain pests via bioacoustic detection. **Crop Protection**, **85**: 71–78.

30. Wang, D., Zhou, H., Yang, H., Shen, Y., Cao, Y., Zhao, H., 2016. Research on Image Acquisition and Recognition for Stored Grain Pests, pp. 268–272, *2016 2nd International Conference on Artificial Intelligence and Industrial Engineering*, Atlantis Press, .

31. Mohammed Othman Farea, S., Kanaan, M., 2019. Support Vector Machine Classifier of Grain Stored in Silo-bags by Using Wireless Network of Temperature and Moisture Sensors, pp. 11–14, *International Congress on Human-Computer Interaction, Optimization and Robotic Applications*, SETSCI, Nevsehir, Turkey.

32. Kanaan, M., Bavkara, C.K., 2018. Proactive Monitoring and Classification of Stored Grain Condition via Wireless Sensor Networks and Machine Learning Techniques, *ISMSIT 2018 - 2nd International Symposium on Multidisciplinary Studies and Innovative Technologies, Proceedings*, Institute of Electrical and Electronics Engineers Inc., .

33. https://archive.ics.uci.edu/ml/index.php. Accessed June 21, 2020.

34. Chen, P., Xie, Y., Jin, P., Zhang, D., 2018. A wireless sensor data-based coal mine gas monitoring algorithm with least squares support vector machines optimized by swarm intelligence techniques. **Special Collection Article International Journal of Distributed Sensor Networks**, **14**(5): 2018.

35. Kayabasi, A., Toktas, A., Sabanci, K., Yigit, E., 2018. Automatic classification of agricultural grains: Comparison of neural networks. **Neural Network World**, **28**(3): 213–224.

36. Sabancı, K., Akkaya, M., 2016. Classification of Different Wheat Varieties by Using Data Mining Algorithms. **International Journal of Intelligent Systems and Applications in Engineering**, **4**(2): 40.

37. Lin, K.P., Chang, H.F., Chen, T.L., Lu, Y.M., Wang, C.H., 2016. Intuitionistic fuzzy C-regression by using least squares support vector regression. **Expert Systems with Applications**, **64**: 296–304.

38. Kowalski, P.A., Kusy, M., 2018. Sensitivity analysis for probabilistic neural network structure reduction. **IEEE Transactions on Neural Networks and Learning Systems**, **29**(5): 1919–1932.

39. Kim, J.H., 2009. Estimating classification error rate: Repeated cross-validation, repeated hold-out and bootstrap. **Computational Statistics and Data Analysis**, **53**(11): 3735–3745.

40. Bengio, Y., Grandvalet, Y., 2004. No unbiased estimator of the variance of K-fold cross-validation. **Journal of Machine Learning Research**, **5**(Sep): 1089–1105.

41. Hsu, C.W., Lin, C.J., 2002. A comparison of methods for multiclass support vector machines. **IEEE Transactions on Neural Networks**, **13**(2): 415–425.

42. Chapelle, O., Vapnik, V., Bousquet, O., Mukherjee, S., 2002. Choosing multiple parameters for support vector machines. **Machine Learning**, **46**(1–3): 131–159.

43. Vapnik, V., Chapelle, O., 2000. Bounds on error expectation for support vector machines. **Neural Computation**, **12**(9): 2013–2036.

44. Keerthi, S.S., 2002. Efficient tuning of SVM hyperparameters using radius/margin bound and iterative algorithms. **IEEE Transactions on Neural Networks**, **13**(5): 1225–1229.

45. Chung, K.M., Kao, W.C., Sun, C.L., Wang, L.L., Lin, C.J., 2003. Radius Margin Bounds for Support Vector Machines with the RBF Kernel. **Neural Computation**, **15**(11): 2643–2681.

46. Ayat, N.E., Cheriet, M., Suen, C.Y., 2002. Optimization of the SVM kernels using an empirical error minimization scheme, pp. 354–369, *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Springer Verlag, .

47. Wang, L., 2008. Feature selection with kernel class separability. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, **30**(9): 1534–1546.

48. Wu, K.P., Wang, S. De, 2009. Choosing the kernel parameters for support vector machines by the inter-cluster distance in the feature space. **Pattern Recognition**, **42**(5): 710–717.

49. Sun, J., Zheng, C., Li, X., Zhou, Y., 2010. Analysis of the distance between two classes for tuning SVM hyperparameters. **IEEE Transactions on Neural Networks**, **21**(2): 305–318.

50. Yin, S., Yin, J., 2016. Tuning kernel parameters for SVM based on expected square distance ratio. **Information Sciences**, **370**–**371**: 92–102.

51. Imbault, F., Lebart, K., 2004. A stochastic optimization approach for parameter tuning of Support Vector Machines, pp. 597–600, *Proceedings - International Conference on Pattern Recognition*, .

52. Xuefeng, L., Fang, L., 2002. Choosing multiple parameters for SVM based on genetic algorithm, pp. 117–119, *International Conference on Signal Processing Proceedings, ICSP*, Institute of Electrical and Electronics Engineers Inc., .

53. Rojas, S.A., Fernandez-Reyes, D., 2005. Adapting multiple kernel parameters for support vector machines using genetic algorithms, pp. 626–631, *2005 IEEE Congress on Evolutionary Computation, IEEE CEC 2005. Proceedings*, .

54. Liu, S., Jia, C.Y., Ma, H., 2005. A new weighted support vector machine with GA-based parameter selection, pp. 4351–4355, *2005 International Conference on Machine Learning and Cybernetics, ICMLC 2005*, .

55. Lorena, A.C., De Carvalho, A.C.P.L.F., 2008. Evolutionary tuning of SVM parameter values in multiclass problems. **Neurocomputing**, **71**(16–18): 3326–3334.

56. Pai, P.F., Hong, W.C., 2005. Support vector machines with simulated annealing algorithms in electricity load forecasting. **Energy Conversion and Management**, **46**(17): 2669–2688.

57. Pai, P.F., Hong, W.C., 2006. Software reliability forecasting by support vector machines with simulated annealing algorithms. **Journal of Systems and Software**, **79**(6): 747–755.

58. Lin, S.W., Lee, Z.J., Chen, S.C., Tseng, T.Y., 2008. Parameter determination of support vector machine and feature selection using simulated annealing approach. **Applied Soft Computing Journal**, **8**(4): 1505–1512.

59. De Souza, B.F., De Carvalho, A.C.P.L.F., Calvo, R., Ishii, R.P., 2006. Multiclass SVM model selection using particle swarm optimization, *Proceedings - Sixth International Conference on Hybrid Intelligent Systems and Fourth Conference on Neuro-Computing and Evolving Intelligence, HIS-NCEI 2006*, .

60. Subasi, A., 2013. Classification of EMG signals using PSO optimized SVM for diagnosis of neuromuscular disorders. **Computers in Biology and Medicine**, **43**(5): 576–586.

61. Xue, Z., Du, P., Su, H., 2014. Harmonic analysis for hyperspectral image classification integrated with PSO optimized SVM. **IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing**, **7**(6): 2131–2146.

62. Raj, S., Ray, K.C., 2017. ECG Signal Analysis Using DCT-Based DOST and PSO Optimized SVM. **IEEE Transactions on Instrumentation and Measurement**, **66**(3): 470–478.

63. Samadzadegan, F., Hasani, H., Schenk, T., 2012. Simultaneous feature selection and SVM parameter determination in classification of hyperspectral imagery using Ant Colony Optimization. **Canadian Journal of Remote Sensing**, **38**(2): 139–156.

64. Li, X., Zheng, A., Zhang, X., Li, C., Zhang, L., 2013. Rolling element bearing fault detection using support vector machine with improved ant colony optimization. **Measurement: Journal of the International Measurement Confederation**, **46**(8): 2726–2734.

65. Raj, S., Ray, K.C., 2018. Application of variational mode decomposition and ABC optimized DAG-SVM in arrhythmia analysis, pp. 1–5, *2017 7th International Symposium on Embedded Computing and System Design, ISED 2017*, Institute of Electrical and Electronics Engineers Inc., .

66. Das, S.P., Achary, N.S., Padhy, S., 2016. Novel hybrid SVM-TLBO forecasting model incorporating dimensionality reduction techniques. **Applied Intelligence**,

45(4): 1148–1165.

67. Das, S.P., Padhy, S., 2018. A novel hybrid model using teaching–learning-based optimization and a support vector machine for commodity futures index forecasting. **International Journal of Machine Learning and Cybernetics**, **9**(1): 97–111.

68. Jiang, M., Luo, J., Jiang, D., Xiong, J., Song, H., Shen, J., 2016. A Cuckoo Search-Support Vector Machine Model for Predicting Dynamic Measurement Errors of Sensors. **IEEE Access**, **4**: 5030–5037.

69. Zhang, X., Wang, J., Zhang, K., 2017. Short-term electric load forecasting based on singular spectrum analysis and support vector machine optimized by Cuckoo search algorithm. **Electric Power Systems Research**, **146**: 270–285.

70. Shaowu, Z., Lianghong, W., Xiaofang, Y., Wen, T., 2007. Parameters selection of SVM for function approximation based on Differential Evolution, pp. 529–535, *International Conference on Intelligent Systems and Knowledge Engineering 2007*, Atlantis Press, .

71. Bhadra, T., Bandyopadhyay, S., Maulik, U., 2012. Differential Evolution Based Optimization of SVM Parameters for Meta Classifier Design. **Procedia Technology**, **4**: 50–57.

72. Elhariri, E., El-Bendary, N., Hassanien, A.E., Abraham, A., 2016. Grey Wolf optimization for one-against-one multi-class support vector machines, pp. 7–12, *Proceedings of the 2015 7th International Conference of Soft Computing and Pattern Recognition, SoCPaR 2015*, Institute of Electrical and Electronics Engineers Inc.

73. Vapnik, V., 2000. The Nature Of Statistical Learning Theory, Second Edition. Springer Science & Business Media, New York, 314 pp.

74. Cortes, C., Vapnik, V., 1995. Support-vector networks. **Machine Learning**, **20**(3): 273–297.

75. Wang, L., 2005. Support Vector Machines: Theory And Applications, First Edition. Springer-Verlag Berlin Heidelberg, Berlin, Heidelberg, 431 pp.

76. Hsu, Chih-Wei and Chang, Chih-Chung and Lin, C.-J. and others, 2003. A practical guide to support vector classification.

77. Lin, H.-T., Lin, C.-J., 2003. A Study on Sigmoid Kernels for SVM and the Training of non-PSD Kernels by SMO-type Methods. **Submitted to Neural Computation**, **3**(1–32): 16.

78. Platt, J.C., 1998. Sequential Minimal Optimization: A Fast Algorithm For Training Support Vector Machines.

79. Dietterich, T.G., Bakiri, G., 1994. Solving Multiclass Learning Problems via Error-Correcting Output Codes. **Journal of Artificial Intelligence Research**, **2**: 263–286.

80. Holland, J.H.., 1992. Genetic Algorithms understand Genetic Algorithms. **Scientific American**, **267**(1): 66–73.

81. Poli, R., Kennedy, J., Blackwell, T., 2007. Particle swarm optimization: An overview. **Swarm Intelligence**, **1**(1): 33–57.

82. Dorigo, M., Di Caro, G., 1999. Ant colony optimization: A new meta-heuristic, pp. 1470–1477, *Proceedings of the 1999 Congress on Evolutionary Computation, CEC 1999*, IEEE Computer Society, .

83. Karaboga, D., Basturk, B., 2007. A powerful and efficient algorithm for numerical function optimization: Artificial bee colony (ABC) algorithm. **Journal of Global Optimization**, **39**(3): 459–471.

84. Wolpert, D.H., Macready, W.G., 1997. No free lunch theorems for optimization. **IEEE Transactions on Evolutionary Computation**, **1**(1): 67–82.

85. Charytanowicz, M., Niewczas, J., Kulczycki, P., Kowalski, P.A., Łukasik, S., 2016. Discrimination ofwheat grain varieties using X-ray images, pp. 39–50, *Advances in Intelligent Systems and Computing*, Springer Verlag, .

86. Jolliffe, I., 2002. Principal Component Analysis, Second Edition. Springer, , 487 pp.

87. Ali, S., Smith-Miles, K.A., 2006. Improved Support Vector Machine Generalization Using Normalized Input Space, pp. 362–371, *Australasian Joint*

Conference on Artificial Intelligence, Springer, Berlin, Heidelberg.

88. KOHAVI, R., 1995. A study of cross-validation and bootstrap for accuracy estimation and model selection. **International Joint Conference on Artificial Intelligence, 1995**, 1137–1143.

89. García, S., Luengo, J., Herrera, F., 2015. Data Preprocessing In Data Mining, First Edition. Springer, , 320 pp.

90. Kruskal, W.H., Wallis, W.A., 1952. Use of Ranks in One-Criterion Variance Analysis. **Journal of the American Statistical Association**.

91. Neuhäuser, M., 2011. Wilcoxon–Mann–Whitney Test, pp. 1656–1658, *International Encyclopedia of Statistical Science*, Springer Berlin Heidelberg, .

92. Derrac, J., García, S., Molina, D., Herrera, F., 2011. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. **Swarm and Evolutionary Computation**.

93. https://www.medcalc.org/manual/chi-square-table.php. Accessed June 25, 2020.

CURRICULUM VITAE

PERSONAL DATA

Name and Surname	:	Shawqi Mohammed Othman Farea
Nationality	:	Yemen
Birth Date and Place	:	14/05/1991 in Yemen-Ibb
Marital Status	:	Single
Tel:	:	+90 5315794994
email	:	4012031611@erciyes.edu.tr shawqialoshari91@gmail.com
Address	:	Mevlana Mah. Sehit Fatih Duman Sk. No:24, 38280 Talas/Kayseri

EDUCATION

Degree	Institute	Date of Graduation
Master's degree	Erciyes University	2017-2020
Turkish Preparation	Erciyes University/ERSEM	2016-2017
Bachelor's degree	Taiz University	2009-2014
High School	Taiz Secondary School	2005-2008

PROFESSIONAL EXPERIENCE

Year	Institute	Mission
2016-present	Yemeni Students' Union (Kayseri/Turkey)	Member
2018-2019	Yemeni Students' Union (Kayseri/Turkey)	Academic Representative
2014-2015	Taiz University	Teaching Assistant

LANGUAGES:

Arabic, English, Turkish

PUBLICATIONS:

1. Mohammed Othman Farea, S., M. Kanaan, 2019. Support Vector Machine Classifier of Grain Stored in Silo-bags by Using Wireless Network of Temperature and Moisture Sensors, pp. 11–14. *International Congress on Human-Computer Interaction, Optimization and Robotic Applications*, SETSCI.