Silesian University of Technology

Faculty of Automatic Control, Electronics and Computer Science

Ensembles of support vector machines with evolutionarily optimized hyperparameters and training sets

This thesis has been submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

mgr inż. Wojciech Dudzik

Supervisor: **dr hab. inż. Michał Kawulok, prof. PŚ**

Assistant supervisor: **dr hab. inż. Jakub Nalepa, prof. PŚ**

Gliwice, March 2023

Contents

Chapter 1

Introduction

Nowadays, the field of machine learning and artificial intelligence (AI) is experiencing significant growth and is considered to be one of the most rapidly evolving fields in computer science. The advent of large-scale data collection and storage, coupled with advancements in computational power, has made it possible to develop machine learning models that are capable of extracting meaningful insights from complex data. As a result, machine learning is now widely used across a range of domains, from healthcare [108] and finance [36] to transportation [140] and entertainment [55]. Despite the significant progress that has been made in the field, the use of these models can still pose challenges to both experienced and inexperienced users. One of the major issues emerges from the large-scale datasets that are often used nowadays. Such datasets can be extremely complex and can be hard to utilize due to high computational costs. As a result, methods for optimizing machine learning models need to be designed to scale to large datasets while maintaining their accuracy and efficiency. Moreover, with a large number of different methods available for performing those tasks, it can be difficult for users to select the most appropriate method for their specific needs.

Classification is one of the fundamental problems in the field of machine learning. It involves the task of predicting the class labels of new data points (samples) based on a model learned on the training dataset of labeled examples. It is a crucial problem in many domains, such as healthcare [132, 103], finance [21], and security [3], where the ability to accurately and efficiently classify samples can provide significant benefits. The classification can be divided into three categories based on the number of classes. There is the one-class classification that tries to identify objects of a specific class among all others. If there are only two classes, the problem is called binary classification while for any larger number, it is described as multiclass classification. Through the years multiple methods were developed to tackle these problems. It leads to the situation where there is a significant number of available optimization techniques and methods to use, including, but not limited to random forest, k-nearest neighbors (k-NN), artificial neural networks (ANNs), and support vector machines (SVMs), can make it difficult for users to determine which approach is best suited for their particular data set. Furthermore, joining multiple models into an ensemble is a widely used technique, which can help to improve the accuracy and robustness of the classifier. The rationale behind ensemble methods is that by combining the predictions of multiple models, we can reduce the variance and bias of the resulting classifier, and thus improve its generalization performance. It, however, brings even more difficulties as there are many different ways of building those ensembles, spanning across bagging [84], stacking [134], and boosting [46] to name the most popular ones. While the problem of appropriate hyperparameters' selection and training of individual models in ensembles still remains to be solved.

SVMs are one of the popular methods for performing binary classification. They work by finding the hyperplane that separates the two classes of data points while maximizing the margin between the hyperplane and closest data points. Those closest points are often selected as support vectors that determine the decision boundary (hyperplane) of the SVM classifier and are crucial for the classification process of new data points. This approach leads to a robust and well-generalizing classifier, which can handle high-dimensional and noisy data [69]. Because of that, they are widely used for binary classification, with a proven track record of high accuracy and robustness in various domains, including bioinformatics [64], finance [79], and engineering [35]. Thanks to that, they are still relevant today in spite of the popularity of deep neural networks (DNNs) and continue to play a critical role in the field of machine learning. While DNNs have revolutionized several fields, including image recognition [27], speech recognition [130], and natural language processing [20], they also have certain limitations. There are new works that propose to join DNNs capabilities in automatic feature extraction with the robustness of SVMs [8] which can provide state-of-the-art results in various domains. However, the major drawback of such solutions is that the SVMs can be computationally expensive to train. This process is known to have time and memory complexities of $O(t^3)$ and $O(t^2)$, respectively, where t is the size of the training set [105]. What is more, the choice of a kernel function can have a significant impact on the performance of the classifier. Hence combining SVMs with DNNs is difficult as DNNs require vast amounts of data for training which tends to produce enormous datasets that could not be handled by regular SVM classifiers. There are indeed numerous approaches toward improving the scaling properties of SVM. These are among other methods that accelerate the training process by better utilizing hardware or providing an implementation that uses the power of graphics processing unit (GPU) [133]. Other works focus on limiting the size of the training set [98] by selecting the most promising vectors hence accelerating the training process and lowering memory requirements. Although the great successes shown in many papers, the majority of these methods ignore the problem of hyperparameters optimization which is crucial in order to get a good classifier. What is more, many of these methods require technical knowledge and proper expertise before use.

Despite the growing popularity of SVMs in machine learning, the optimization of these models remains a complex and challenging task. The large size of datasets often necessitates lengthy computations and requires a vast amount of memory, which can impede the effective training of SVMs in practical applications. In addition, the problem of hyperparameter optimization and the selection of relevant features are often omitted or addressed independently of each other which may negatively affect the performance of the SVM classifier. This dissertation aims to propose effective solutions for optimizing SVM models in the context of binary classification. These solutions have been designed to overcome the challenges posed by large datasets and provide efficient approaches for hyperparameter optimization training set and feature selection problems simultaneously. Furthermore, the dissertation introduces new methods for constructing ensembles of SVM models to enhance their performance and extend their capabilities with special attention put on keeping the training process fast. In order to achieve all of the goals evolutionary computations are used which already proved to be a robust solution to many problems while some examples of such techniques were already effectively utilized to optimize SVMs. This work is built upon the success of those methods and improves them. By developing these new techniques, this dissertation seeks to advance the state-of-the-art SVM optimization.

In the next part of this chapter, the research hypotheses are stated. Then, Section 1.2 provides a summary of papers that are published which contain the methods described in this dissertation. These methods have already been published by the author in proceedings of international conferences and one in a peer-reviewed journal. Finally, the last section provides a roadmap to the remaining part of this dissertation.

1.1 The research hypotheses

There are two research hypotheses formulated for this work:

1. Simultaneous optimization of the training set and the SVM hyperparameters improve training and classification time compared to other state-of-the-art methods proposed for this purpose without affecting the classification quality.

2. SVM ensembles created using evolutionary algorithms provide improved classification performance compared to other well-established methods, including existing algorithms for building SVM ensembles.

To clarify what is meant by "other state-of-the-art methods proposed for this purpose" in the first hypothesis—these methods comprise grid search and optimization algorithms for SVM hyperparameters, and training set selection methods described in the existing literature. It is understood that these methods yield comparable classification metric values to the proposed algorithm while providing faster training and classification time. This final effect on the classification performance is determined by statistical tests comparing multiple classification metrics. The second hypothesis aims to measure the improvement in classification performance using classification accuracy, F1, and Matthews correlation coefficient (MCC) score (as described in Section 2.1.4). Other aspects such as training and classification time will also be analyzed for all of the approaches as they are important for practical applications. This work also includes the algorithms for feature selection but they are not studied in detail and should be treated as an extension to the training set selection and hyperparameters optimization.

1.2 Published papers

Parts of the work presented in this dissertation have already been published in international conferences and in peer-reviewed journals. Table 1.1 shows all of the published papers.

Table 1.1: The list of proposed algorithms that are already published in international conferences and peer-reviewed journals.

Algorithm	Acronym	Reference
ALternating **G**enetic **A**lgorithm for selecting SVM training sets and models	ALGA	[76]
ALternating **M**emetic **A**lgorithm for selecting SVM training sets and models with **F**eature **S**election	FSALMA	[39]
Evolutionarily-tuned **SVMs**	ESVM	[37]
Simultaneously-**E**volved **SVMs**	SE-SVM	[38]
SVMs with **A**daptive **RBF** kernels	ARBF-SVM	[94]
Data-**A**daptive **SVM**	DA-SVM	[40]
Cascades of **E**volutionarily optimized **SVM**s	CE-SVM	[41]

It is important to note that the publications listed in Table 1.1 are not the only papers co-authored by the author of this dissertation. Rather, these works represent key contributions that led to the development of algorithms for optimizing SVMs. Any figures that have already been published in one of these works will be clearly indicated in the figure's caption, e.g., "This figure comes from our paper [37]".

1.3 Structure of this dissertation

This work is organized as follows. Chapter 2 provides the background concerning the theory behind SVMs. Here, the importance of hyperparameters in SVM is discussed and presented using visual examples of 2D datasets. Additionally, a sub-section of the chapter addresses the challenge of evaluating the performance of binary classifiers. The second part of the chapter undertakes a comprehensive review of the existing literature on SVMs. The review is presented in subsections based on the optimization aspects of SVM, including the SVM model, training set, and feature set. In addition, the process of building ensembles, a technique that involves combining several classifiers to enhance performance, is analyzed. Toward the end of the chapter, a summary of all the methods discussed is presented, and their relevance in the current research context is highlighted. This summary consolidates the various approaches to SVM and helps better contextualize the presented work within the broader context of the current research.

Chapter 3 is dedicated to the presentation and analysis of newly proposed solutions. The chapter provides a detailed description of each of the algorithms, outlining their individual features and functionalities. The objective of this chapter is to offer a comprehensive understanding of the solutions, including the rationale behind their design and implementation. Finally, a short summary is presented, which compares the algorithms. This comparison provides readers with a better overview of the provided solutions and offers a basis for understanding the relevance of each of the solutions in addressing the research problem.

Chapter 4 presents an experimental study aimed at assessing the performance of the methods discussed in Chapter 3. The chapter offers a detailed description of the datasets used in the experiments and outlines the configurations used for the algorithms. The objective of this chapter is to provide a thorough analysis of the proposed algorithms and to evaluate their performance. These new algorithms are subjected to a quantitative analysis using 2D artificially created datasets. This analysis is aimed at examining the behavior of the algorithms and enables to compare them visually. Furthermore, the performance comparison of the proposed methods against other state-of-the-art optimization techniques and popular classifiers, using benchmark datasets is presented. This comparison is critical in evaluating the effectiveness of the new methods and in determining their contributions to the field. The statistical significance of the results is also discussed, ensuring that the conclusions drawn from the study are based on reliable evidence.

Chapter 5 of this dissertation summarizes the finding and provides the conclusions drawn from the presented data. This chapter encapsulates the key takeaways from the study, highlighting the significance of the findings and how they contribute to the existing knowledge in the field. Furthermore, the chapter explores potential avenues for future research, sheds light on the limitations of the study, and identifies areas that require further exploration.

Chapter 2

Theory and literature review

In this chapter, the theory behind SVMs is introduced. Hard margin and soft margin SVMs are discussed (in Section 2.1.1), which focus on linear SVMs. The non-linear SVMs (using kernel functions) are presented separately in Section 2.1.2. The significance of using the appropriate kernel along with its hyperparameters is discussed in Section 2.1.3. The examples showing the impact of those kernels will be presented with 2D datasets and visualized. Later in the chapter, the metrics used to evaluate the performance of SVMs are presented (in Section 2.1.4). This is followed by a brief introduction to the evolutionary computations which are used in all of the proposed methods. This concludes the first part of this chapter. The second part of this chapter is dedicated to the literature review (Section 2.3). This review is divided into multiple sections, concerning the methods for SVM hyperparameters optimization (Section 2.3.1) followed by methods for training set selection (Section 2.3.2) and feature selection (Section 2.3.3). Afterwards, methods that couple any of those optimizations together are discussed in Section 2.3.4. The chapter is concluded with the review of the approaches toward buildings ensembles with the SVMs in Section 2.3.5.

2.1 Support vector machines

SVM is a binary supervised classification algorithm. It was introduced by V. Vapnik et al. in 1992 [19] and later extended for regression tasks. The fundamental idea of SVM is to find the optimal separating hyperplane between the two classes that maximizes the margin between the closest data points from each class.

Assuming a set $\boldsymbol{T}$ that consists of t training vectors $x_i \in \mathbb{R}^D, i = 1, ..., t$, where each vector belongs to one of two classes $y_i \in \{-1, +1\}$ (binary classification problem). A function $f(x)$ separates those vectors into two classes. This function can be defined as:

$$\begin{aligned} f(\mathbf{x_i}) > 0, \forall y_i = 1, \\ f(\mathbf{x_i}) < 0, \forall y_i = -1. \end{aligned} \tag{2.1}$$

First, the problem of determining $f(x)$ on linearly separable data is considered.

2.1.1 Hard margin SVMs

Linear SVM classifier aims to separate the data in the D-dimensional input space using a hyperplane decision boundary defined as:

$$f(x) : \mathbf{w}^T\mathbf{x} + b = 0, \tag{2.2}$$

where $\mathbf{w}$ is a hyperplane normal vector $\mathbf{w} \in \mathbb{R}^D$, and b is an offset, $b \in \mathbb{R}$. This hyperplane Equation needs to satisfy the conditions defined in Equation 2.1. There might be many hyperplanes equations that will satisfy those conditions. To determine an unambiguous solution, this decision hyperplane is positioned such that the distance between two vectors from opposite classes is maximal (with respect to the hyperplane). Considering data that are linearly separable and contain only two classes ($y_i \in \{-1, +1\}$), training data need to satisfy the following conditions:

$$\begin{aligned} \mathbf{w}^T\mathbf{x_i} + b \geq 1, y_i = +1, \\ \mathbf{w}^T\mathbf{x_i} + b \leq -1, y_i = -1, \end{aligned} \tag{2.3}$$

which can also be written in a single Equation as:

$$y_i(\mathbf{w}^T\mathbf{x_i} + b) - 1 \geq 0, y_i \in -1, +1. \tag{2.4}$$

The inequalities shown in Equation 2.3 and Equation 2.4 present two parallel hyperplanes (i.e., hyperplanes with the same normal vector) which are based on vectors that satisfy following equation:

$$\begin{aligned} \mathbf{w}^T\mathbf{x_i} + b &= 1, \\ \mathbf{w}^T\mathbf{x_i} + b &= -1. \end{aligned} \tag{2.5}$$

These vectors are called support vectors (SVs). The distance to the origin is expressed as $\frac{|1-b|}{||\mathbf{w}||}$ and $\frac{|-1-b|}{||\mathbf{w}||}$ respectively, where $\|\mathbf{w}\|$ is the second norm of a vector (called also Euclidean norm). Those planes are presented with dotted lines in Figure 2.1. Please note that with the above definition, there are no vectors in space between those hyperplanes. What is more, the shortest distance to any data vector (from either class) is $\frac{1}{\|\mathbf{w}\|}$, hence the maximal margin (m) is equal to:

$$m = \frac{2}{\|\mathbf{w}\|}. \tag{2.6}$$

The example of such linear SVM is presented in Figure 2.1b. Moreover, if any of the selected SVs in Figure 2.1b is changed to another vector from the training set, the calculated solution (hyperplane) will be different.

As the goal is to maximize the separating margin, the value of $||\mathbf{w}||$ needs to be minimized:

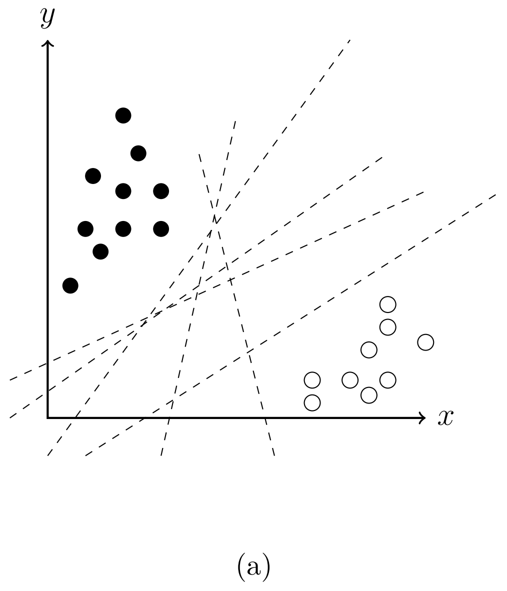
$$\min_{\mathbf{w},b} \|\mathbf{w}\|. \tag{2.7}$$

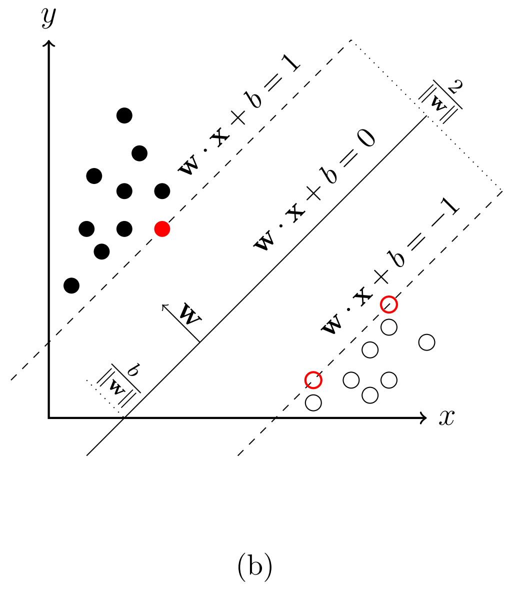
To simplify the calculation, we can rewrite Equation 2.7 as the quadratic term:

$$\min_{\mathbf{w},b} \frac{\|\mathbf{w}\|^2}{2}. \tag{2.8}$$

While solving the optimization problem, we need to respect the constraints from Equation 2.3. Therefore, this becomes a quadratic programming problem (QP). The solution obtained by solving such formulation (called also primal form) is a hyperplane defined by the normal vector which is retrieved from QP. This hyperplane is then used to classify incoming data based on the following decision function:

$$f(a) = sgn(\mathbf{w}^T\mathbf{a} + b), \tag{2.9}$$





y
x
$\mathbf{w}\cdot\mathbf{x}+b=1$
$\mathbf{w}\cdot\mathbf{x}+b=0$
$\mathbf{w}\cdot\mathbf{x}+b=-1$
$\frac{2}{\|\mathbf{w}\|}$
$\frac{b}{\|\mathbf{w}\|}$
$\mathbf{w}$
(b)

Figure 2.1: (a) An illustrative example of simple 2D datasets with two classes (black and white dots) where multiple separating hyperplanes are denoted with dotted lines. (b) An example of a linear SVM trained on the presented data. The support vectors are denoted with red color, while the decision boundary is presented with a solid line.

where **a** is a data vector.

Equation 2.3 can be re-written to obtain the Lagrangian in its primal form:

$$\mathcal{L}(\mathbf{w}, b, \alpha) = \frac{\|\mathbf{w}\|^2}{2} - \sum_{i=1}^{t} \alpha_i y_i (\mathbf{w}^T \mathbf{x_i} + b) + \sum_{i=1}^{t} \alpha_i, \tag{2.10}$$

where α_i are the Lagrange multipliers. The advantage of such formulation is the fact that all of the training and test data will appear in the form of a dot product between vectors. This is a vital feature that will allow the expansion of the above formulation for non-linear cases. Finding the hyperplane for SVMs is equivalent to finding the solution to the Karuch-Kuhn-Tucker (KKT) conditions. The Lagrangian in the dual form, also known as Wolfe dual, can be written by employing those conditions to obtain the following form:

$$\mathcal{L}_D(\alpha) = \sum_{i=1}^{t} \alpha_i - \frac{1}{2} \sum_{i=1}^{t} \sum_{j=1}^{t} \alpha_i \alpha_j y_i y_j \mathbf{x_i^T} \mathbf{x_j}, \tag{2.11}$$

under the following constraints:

$$\sum_{i=1}^{t} \alpha_i y_i \mathbf{x_i} = 0, \alpha_i \geq 0. \tag{2.12}$$

The weight vector of the original problem is expressed as:

$$\mathbf{w} = \sum_{i=1}^{t} \alpha_i y_i \mathbf{x_i}. \tag{2.13}$$

The decision function from Equation 2.9 can be obtained by applying Equation 2.13:

$$f(\mathbf{a}) = sgn(\sum_{i=1}^{t} \alpha_i y_i \mathbf{x_i}^T \mathbf{a} + b). \tag{2.14}$$

Soft margin SVMs

In 1995, a method for learning SVMs with a soft margin was published [32]. It is a modification of the SVM classifier that allows the determination of a decision function for non-linearly separable data (see Figure 2.2). Such data often occur in real-world problems due to, among other things, noisy readings or incorrect labeling of training data. Otherwise trying to train hard-margin SVM on noisy data will complicate the boundary decision boundary function, as well as degrade the quality of the classification itself. In order to be able to apply inequalities 2.3, an additional variable must be introduced, which is the cost of the vector misclassification operation:

$$\begin{aligned} \mathbf{w^T x_i} + b \geq 1 - \xi_i, y_i = +1, \\ \mathbf{w^T x_i} + b \leq -1 + \xi_i, y_i = -1, \\ \xi_i \geq 0, \end{aligned} \tag{2.15}$$

where ξ_i is a slack variable. By taking the slack variable into consideration in the objective function, it can be updated as follow:

$$\min_{\mathbf{w},b,\xi} \frac{\|\mathbf{w}\|^2}{2} + C \sum_{i=1}^{t} \xi_i, \tag{2.16}$$

under the following constraints:

$$\begin{aligned} y_i(\mathbf{w^T x_i} + b) \geq 1 - \xi_i, i = 1, ..., t, \\ \xi_i \geq 0, i = 1, ..., t, \end{aligned} \tag{2.17}$$

where C is the parameter that controls compromise between the margin and the slack penalty. The larger values of C increase the penalty for errors. This new parameter allows for presenting the concept of soft margin SVMs. Similarly, as in the previous case, the Equation 2.11 can be re-written for soft margin SVM:

$$\mathcal{L}_D(\alpha) = \sum_{i=1}^{t} \alpha_i - \frac{1}{2} \sum_{i=1}^{t} \sum_{j=1}^{t} \alpha_i \alpha_j y_i y_j \mathbf{x_i^T x_j} \tag{2.18}$$

is to be maximized subject to

$$\begin{aligned} &\sum_{i=1}^{t} \alpha_i y_i = 0, \\ &0 \leq \alpha_i \leq C. \end{aligned} \tag{2.19}$$

The weight vector $\mathbf{w}$ is the same as in Equation 2.13. In comparison to the previous problem statement, the α_i parameters are additionally bounded by the C value. For very large values of C, this formulation is close to the hard margin SVM in practice. To better illustrate the concept situation of misclassification of a single vector is presented in Figure 2.2. One of the white vectors is on the "wrong" side of the decision hyperplane, therefore, there is a penalty to be accounted for. For every *i-th* vector, this penalty is equal to $d = \frac{\xi_i}{|\mathbf{w}|}$.

2.1.2 Non-linear SVMs

While the above description of linear SVMs can be used for classification, there are still many real-life problems where a non-linear decision function is required. For this reason, the *kernel trick* was introduced to provide a non-linear separating hyperplane in SVMs [32]. It includes defining a kernel function which is computed as a dot product of two vectors:

$$K(\mathbf{a_i}, \mathbf{a_j}) = \phi(\mathbf{a_i})^T \phi(\mathbf{a_j}), \tag{2.20}$$

where $\mathbf{a}$ denotes the sample (vector) from the input space, and $\phi : \mathbb{R} \rightarrow \mathbb{F}$ is a

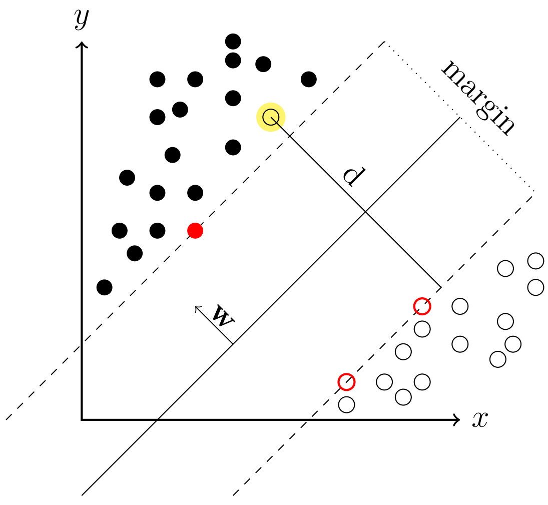


Figure 2.2: An example of the soft-margin linear SVM. The misclassified example is denoted with a yellow circle.

function that maps the data from the original D-dimensional space into a higher-dimensional space $\mathbb{F}$ and $K : \mathbb{R}^D \times \mathbb{R}^D \rightarrow \mathbb{R}$. The kernel function needs to satisfy the Mercer Theorem [62]. This allows for writing Equation 2.11 as:

$$\mathcal{L}_D(\alpha) = \sum_{i=1}^{t} \alpha_i - \frac{1}{2} \sum_{i=1}^{t} \sum_{j=1}^{t} \alpha_i \alpha_j y_i y_j \mathbf{K}(\mathbf{x_i^T}, \mathbf{x_j}), \tag{2.21}$$

thus the decision function used for classification can be written as:

$$f(a) = sgn(\sum_{i=1}^{t} \alpha_i y_i \mathbf{K}(\mathbf{x_i^T}, \mathbf{a}) + b). \tag{2.22}$$

When the kernel function satisfies Mercer conditions, it can be proven after [32] that the dot product is defined in a feature space. Because of that, there is no need to calculate ϕ mapping explicitly. Therefore, kernel trick allows the SVM to operate in the original, lower-dimensional space while still taking advantage of the separation of the data in the higher-dimensional space. It is worth mentioning that in order to classify new vectors there is only a need to calculate the values of a kernel function between support vectors (SVs) and new data sample.

There are several types of kernel functions, including linear, polynomial, and radial basis function (RBF) kernels. The choice of a kernel function will depend on the characteristics of the data and the specific requirements of the task at hand. While this work focuses on RBF and linear kernels (being the most popular ones), there are many more kernels of which a few examples are shown in Table 2.1:

Table 2.1: Popular kernel functions.

Name	Function
Linear	$K(\mathbf{x_i}, \mathbf{x_j}) = \mathbf{x_i}^T\mathbf{x_j}$
Polynomial	$K(\mathbf{x_i}, \mathbf{x_j}) = (\mathbf{x_i}^T\mathbf{x_j} + g)^d$
Radial basis function (RBF)	$K(\mathbf{x_i}, \mathbf{x_j}) = e^{-\gamma\|\|\mathbf{x_i}-\mathbf{x_j}\|\|^2}, \gamma > 0$
Sigmoid	$K(\mathbf{x_i}, \mathbf{x_j}) = tanh(\mathbf{x_i}^T\mathbf{x_j} + g)$

The example of the mentioned mapping and learned hyperplane is presented in Figure 2.3 and 2.4. The image in Figure 2.3 presents the data that cannot be separated linearly. Hence, a proper kernel trick must be used to effectively apply an SVM to this problem. After learning a hyperplane, which is presented with the solid blue line, between classes can be obtained. Figure 2.4 is an example of possible mapping the 2D data into 3D space by using the following formula to obtain the third dimension of the data $z = x^2 + y^2$. After applying that mapping, a linear SVM could be trained which will result in the presented hyperplane (a blue plane). Thanks to the kernel trick this explicit mapping is not necessary and by applying the proper kernel function the same result could be obtained.

2.1.3 Kernel selection

In the context of SVMs, the kernel function plays a crucial role in the performance of the model. As mentioned previously, the kernel function is used to map the input data into a higher-dimensional feature space, where it becomes possible to find a hyperplane that can linearly separate the data. The choice of the kernel function can significantly impact the model's accuracy, as well as the computational efficiency of the training process [54]. One problem with selecting the kernel function for an SVM is that there is no general way to determine the "best" kernel function for a given dataset. Different kernel functions may perform better on different types of data, and it is often necessary to try out a number of different kernel

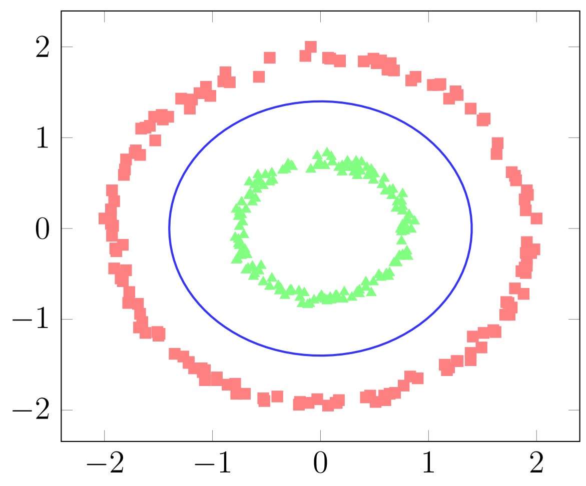


Figure 2.3: An example of data that is non-linearly separable. Green triangles present a positive class while red squares present a negative class. The hyperplane is rendered with a blue solid line.

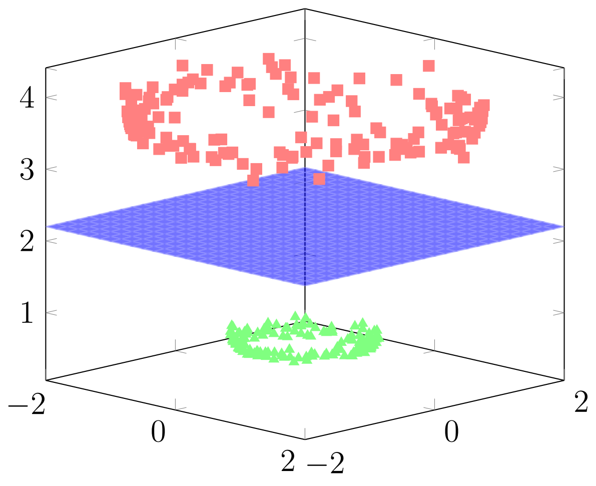


Figure 2.4: One of possible mapping of data from Figure 2.3 into 3D space. Here the decision boundary is seen as a blue 3D plane.

functions in order to find the one that works best for a particular dataset. This can be time-consuming and may require significant experimentation [9]. Another issue with kernel function selection is that it can be difficult to determine the appropriate kernel function for a given dataset without a strong understanding of the characteristics of the data and the underlying relationships between the data points. Moreover, many kernels come with a set of hyperparameters which need to be tuned. This often requires performing cross-validation which also requires time and computing power. This can make it challenging for researchers to effectively use SVMs, particularly if they work with large or complex datasets.

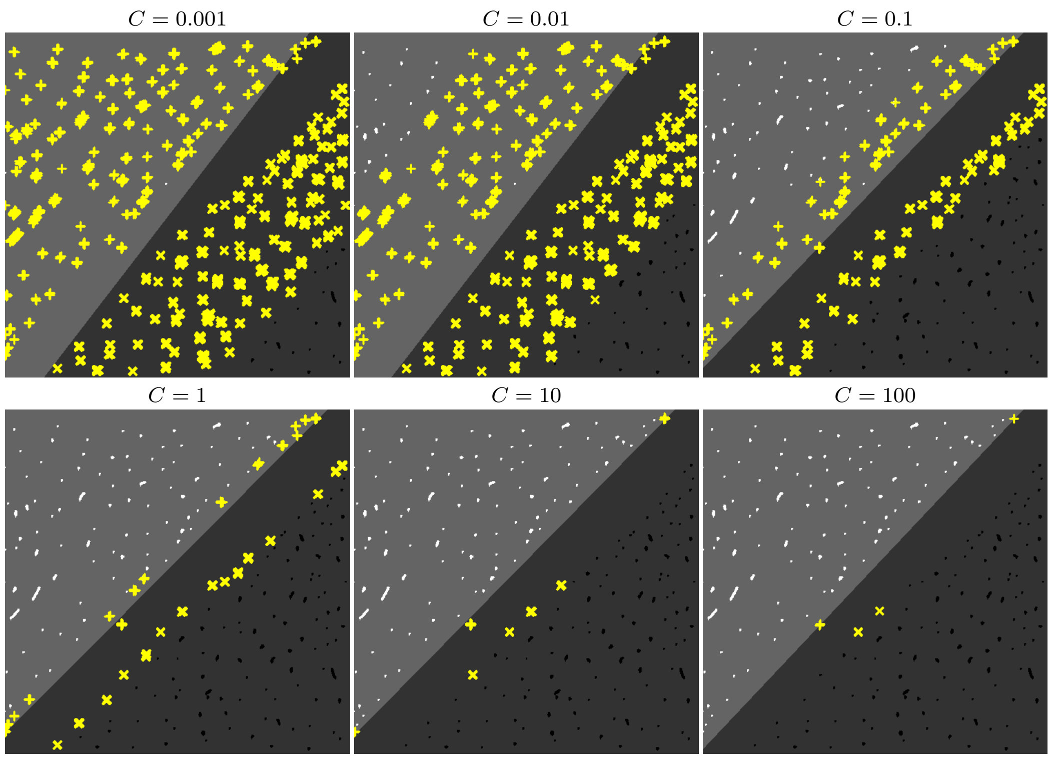


Figure 2.5: Comparison of different C hyperparameter setting for the linear kernel. Yellow crosses denote SVs, whereas dark and light gray denote values of decision function – the boundary between them is the decision hyperplane.

In the following examples, there is the presentation of how selecting a proper kernel function as well as its hyperparameters can affect the final decision hyperplane. For the purpose of better understanding, all of the results of SVMs and their decision boundaries are visualized on artificially created 2D datasets. First, the importance of the C hyperparameter with a linear SVM is presented in Figure 2.5. Large values of C result in a smaller-margin hyperplane to avoid any misclassifications. Conversely, a very small value of C cause a larger margin separating the hyperplane, even if that hyperplane misclassifies more points. This effect can be seen in Figure 2.5, where increasing of the C width of the margin is getting smaller which results in a smaller number of support vectors as they are not lying within this margin and effectively do not influence decision function. In this example, the value of C does not affect the generalization performance of an SVM but it is visible that the selection of the proper value can greatly affect the resulting SVM model. In this case, it impacts the number of SVs hence the classification speed will differ significantly between tested values.

The second example presented in Figure 2.6 compares different SVM kernels and their hyperparameters. The first row presents the example of a 2D dataset (please note that dots were enlarged in the visualization of the dataset) and two very similar solutions obtained by the polynomial and RBF kernel. What can be noticed is that in both of those examples, there are very few SVs present. On the other hand, the second row provides examples of poor classification (in comparison to the best solutions obtained). Starting from the left side, the linear kernel is inappropriate for this kind of data as it requires a non-linear decision boundary. The RBF kernel with wrongly optimized γ and C values drastically increases the number of SVs while achieving an accuracy of 0.94. As the last example, the polynomial kernel with $d = 2$ and $C = 0.001$ provides even worse classification than the linear kernel. These are only a few examples taken from optimizing those sets of hyperparameters for a given kernel using the grid search algorithm. Even in the case of a fairly simple 2D dataset (where perfect classification is easy to achieve), the values of hyperparameters and kernel selection are crucial in order to achieve good classification performance.

2.1.4 Performance evaluation

A typical way to assess the performance of SVMs (as well as other binary classifiers) is to calculate the confusion matrix, presented in Table 2.2 and derive various metrics from it. There are four numbers that create such a matrix: (1) the

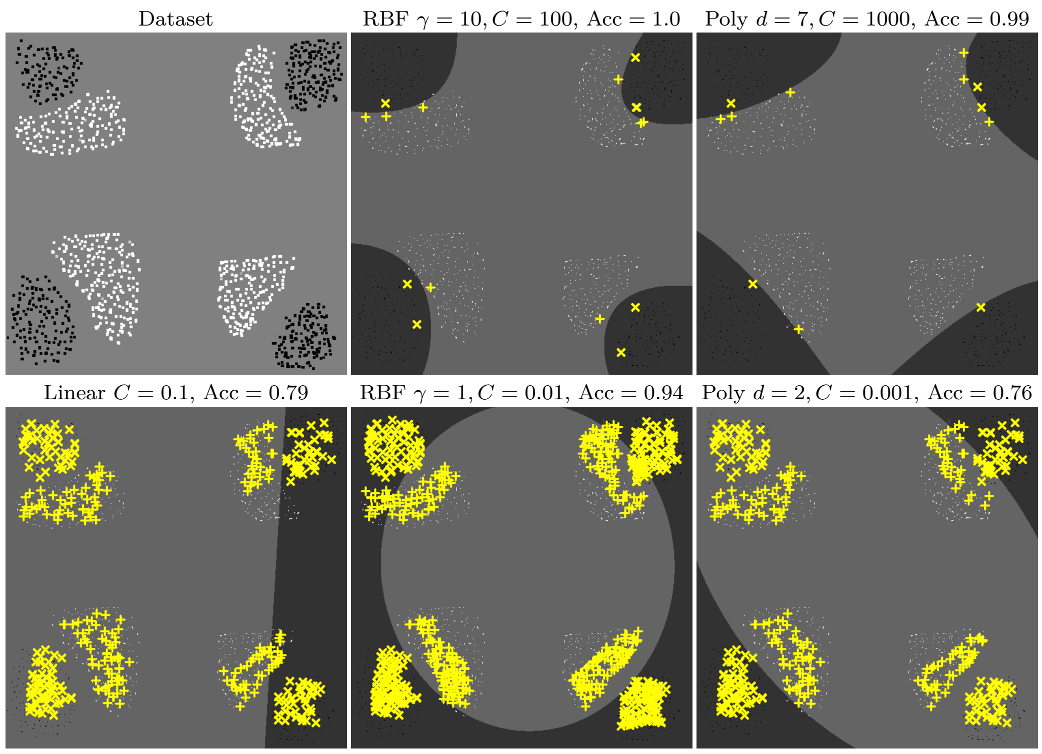


Figure 2.6: Comparison of different SVM kernels (linear, RBF, and polynomial) and their hyperparameters. Yellow crosses denote SVs, whereas dark and light gray denote values of decision function – the boundary between them is the decision hyperplane. Above each example, the kernel with a set of hyperparameters is given together with the accuracy of the model.

Table 2.2: Visualization of confusion matrix, where green cells represent correct classification and red cells represent different types of errors.

Predicted →	**Positive**	**Negative**
Actual ↓		
Positive	True positive	False negative
Negative	False Positive	True negative

number of correctly classified vectors from the positive class called true positive (TP), (2) the number of correctly classified vectors from the negative class called true negative (TN), (3) the number of incorrectly classified vectors from the positive class called false negatives (FN) also known as type II error and finally (4) the number of the incorrectly classified vectors from the negative class called false positive (FP) also known as type I error. Multiple metrics can be derived using those four numbers, such as:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}, \tag{2.23}$$

$$Precision = \frac{TP}{TP + FP}, \tag{2.24}$$

$$Recall = \frac{TP}{TP + FN}, \tag{2.25}$$

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} = \frac{2 \cdot TP}{2 \cdot TP + FN + FP}, \tag{2.26}$$

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}. \tag{2.27}$$

In this dissertation, all of those values will be presented to compare the performance of SVMs. While all of the above metrics can describe performance, not all of them can be a good choice, e.g., for imbalanced data, it can be easy to achieve very high accuracy by always predicting the same (majority) class. For such cases F1 is a better choice for imbalanced data. Although in the case of the F1 score selecting a minority class as a positive one is still important. Despite the crucial importance of selecting proper performance metrics in machine learning, there is no widespread consensus on unified measures. Lately, MCC was proposed as being more reliable than accuracy or F1 [29, 30]. One more important aspect to consider is the decision threshold. Naturally, SVMs decide to which class an example belongs based on the sign of the decision function. This value could be adjusted based on the receiver operating characteristic (ROC) curve [44]. This process of threshold adjustment is important when SVM is trained using the training set selection procedure. It may seem counterintuitive at first, as there is a free term b (see Equation 2.22) which is found during the optimization process. However, when only a subset of the training data is used to derive a model of an SVM a part of the information about data characteristics is lost. Due to that adjusting threshold based on the full training set or validation set can boost the SVM performance. The training set selection makes SVM training feasible for large datasets but can negatively affect its final performance. Therefore, the selection of an additional threshold based on ROC can impact performance positively as the learned term b cannot know the full characteristic of underlying data. An argument to use the area under the ROC curve (AUC) as the performance metric can be made as it does not require setting certain thresholds but it becomes harder to use in an ensemble setting which is also studied in this work. These difficulties include selecting micro or macro averages of AUC of models that are included in the ensemble which have a different interpretation and their selection depends on the dataset. What is more, the presented method (in Section 3.6) builds a cascade of SVMs where it is not trivial to define the AUC metric and its interpretation. That is why for building ensembles MCC is used as a primary metric (although as presented in Section 3.6, the fitness function for evolutionary computation does not use MCC at all).

Moreover, as a time performance measure for SVM the number of SVs can be retrieved, as the classification time depends linearly on the number of SVs. Also, the training time of the classifier could be monitored. Although measuring the training time depends heavily on the implementation of the training algorithm and its usage of advanced CPU features as well as usage of GPU or memory layout of data used during the training. The observed timing differences could be on the order of hundreds of times [133]. That is why training times are compared mainly among proposed algorithms and not directly compared to others methods as there is a great possibility for non-informative or misleading results. To sum up, non-functional aspects of SVMs that can be measured include:

1. number of SVs,
2. classification time,
3. training time.

2.2 Evolutionary algorithms

Evolutionary algorithms (EAs) are a class of optimization techniques that are inspired by the process of natural selection. These algorithms are used to solve a wide range of problems from different fields such as economics, management and engineering [74]. They are particularly well suited for complex problems or ones with large search spaces that are too difficult to solve using traditional optimization methods, or for which there is no known algorithm [89]. There are multiple branches of EA such as genetic algorithms (GAs), evolution strategies, evolutionary programming, differential evolution, or neuroevolution.

GAs work by iteratively improving a population of candidate solutions to a problem, through the application of a set of rules that mimic the process of natural selection. Each element in such a population is called a chromosome which is the encoded representation of a solution. Those chromosomes are evolved using specific genetic operators. In the basic GA, those operators include crossover and mutation. The crossover is the process of combining two solutions to create a new solution. It may also involve additional checks if the new solution is feasible. The mutation consists of randomly altering a solution to diversify the search. After such alteration, the fitness value of each individual is assessed and it can be either minimized or maximized depending on the GA design and problem at hand. Later fitness values are used to select individuals for the next generation and creation of new population. Most often selection process keeps the best solution while the rest of the population can be selected in different ways. One of the popular choices is to use a strategy that keeps the N best individuals to maintain the constant size of the population. This process is continued until the stop condition is met. The termination condition is an important step in designing the algorithm and it may include:

- checking the fitness of the best individual in the population (whether the individual fitness is satisfying threshold set before running GA),
- checking the number of generations evolved or time of execution,
- checking the population characteristic (e.g., diversity, if average fitness of population has changed compared to the last generation),
- or checking combinations of the above conditions.

The pseudocode of a general GA is presented in Algorithm 1.

Algorithm 1 A general scheme of a genetic algorithm (GA).

```
 1: Initialize population $P_a$
 2: Calculate the fitness of each chromosome in $P_a$
 3: while stop condition not met do
 4:     Select pairs of chromosomes $(p_a, p_b)$ from $P_a$
 5:     for all pairs $(p_a, p_b)$ do
 6:         $p_c$ = crossover($p_a$,$p_b$)
 7:         $p_c$ = mutation($p_c$)
 8:         Calculate fitness of $p_c$
 9:         Insert $p_c$ into $P_b$
10:     $P_a$ = Selection($P_a$,$P_b$)
11: return fittest chromosome in $P_a$
```

When designing the genetic algorithm the choice of a proper chromosome structure (encoding) and fitness metric is crucial. These are often problem dependent parts of GAs. Representations of an individual are chosen so that its set of genes reflects the solution to the problem. For example, in the travelling salesman problem (TSP), a good representation would be an ordered list of towns to visit [131]. Since the GAs operate on a population of solutions, the search starts from many points scattered across the solution space. This makes GA less likely to get the solution stuck in local minima. It is one of the significant advantages of GAs compared to other methods that operate on a single solution. The fitness function has the task of calculating how close a particular individual is to the optimal (sought) solution. Again referring to the TSP, the evaluation function can be defined as the sum of the distances between the cities on the list.

Despite their successes in solving a wide range of various optimization problems [90], evolutionary algorithms do have some limitations. One of the main challenges with these algorithms is their computational complexity. Evolutionary algorithms typically require a large number of function evaluations in order to find good solutions. In addition, evolutionary algorithms may not always find the global optimum solution to a problem and may be sensitive to the initialization of the population [115].

Another problem often encountered in literature is proving the effectiveness of GAs in an analytical way. The main reason for this difficulty is the “No free lunch” theorem [135], stating that there is no optimization algorithm that will be general and simultaneously optimal for solving any problem. For this reason, if no assumptions are made about the objective function f, it is difficult to prove the superiority of evolutionary methods over other methods, e.g., random search. A schema theorem was proposed [63] which says that short, low-order schemata (a set of few genes) with above-average fitness increase exponentially in frequency in successive generations. This means that if there are some genes that present better solutions they should be quickly replicated to other chromosomes in the population. The problem is that this theorem holds under the assumption of GA maintaining an infinitely large population and does not always carry over to populations of the defined (and limited) size used in practice. Due to sampling errors GAs may converge on schemata that have no selective advantage. In practice, empirical evidence is accepted as proof of genetic algorithms’ advantage over other methods [89].

One more important addition (and improvement) to GAs is the use of local search or other refinement procedures. Such an algorithm is called a memetic algorithm (MA) or a hybrid GA [92]. This improvement over the presented general GA structure provides a better balance between generality and problem specificity. It also addresses the “No free lunch” theorem since MA is tailored for the problem with additional refinement procedures. It can be applied to all of the individuals in the population, as well as try to select the most promising chromosomes. As it tries to improve the search result, it also balances the exploration versus exploitation of promising regions of the solution space and helps to avoid premature convergence and getting stuck in local minima. Due to their advantages, MAs have been proposed for a wide range of applications such as machine learning [93], bioinformatics and medicine [12], engineering [47, 85] and combinatorial optimization [91, 123].

2.3 Literature review

In the following section, the state-of-the-art methods for SVM hyperparameters optimization and the training set, and feature set selection are reviewed. Afterwards, the literature regarding the optimization of multiple aspects is analyzed.

2.3.1 Optimization of SVM hyperparameters

Optimization of SVM hyperparameters is an active research topic. In more general terms, it is a part of various automatic hyperparameter optimizations (HPOs) These include simple techniques such as grid and random search as well as more advanced ones like Bayesian optimization or evolution strategies. Running HPO is not a simple task as it is often viewed as a black-box optimization problem and often requires proper configuration which may affect both performance of the HPO algorithm and the quality of results [17]. It is worth noting that such black-box approaches often do not have closed-form mathematical representation, hence there is no analytic gradient information available. What is more, the machine learning models include continuous, categorical, discrete, and conditional hyperparameters which make traditional numerical optimization methods unsuitable for HPO. Another basic and crucial difficulty in such optimization is that in the case of SVMs and other supervised learning methods, the training procedure needs to be run (meaning running optimization process for finding $\|\mathbf{w}\|$ in case of SVMs, see Equation 2.21) before measuring the objective function. This process is known to be problematic for SVM in particular due to time and memory complexities [105]. Taking those properties together they define a considerably difficult optimization problem.

There are several main categories into which the hyperparameter optimization algorithms can be divided:

1. general methods for hyperparameter optimization—these methods do not impose any prior knowledge of the SVM structure,
2. methods focused on kernel function selection and determination of its hyperparameters—often focused on several predefined kernels,
3. specialized methods that create new hand-crafted kernel functions.

As already discussed, the general methods should perform on average worse than the specialized ones (due to the “No free lunch” theorem), but they have the advantage of much easier setup and application which may favor their choice in practice. The presented categories are not exhaustive, as there are far more attributes that can be taken into consideration, such as the exploration vs. exploitation trade-off (global vs. local behavior), parallelizability, noise handling or multi-fidelity (e.g., using subsets of data for quicker evaluation) [138]. Although sub-sampling the training set is used in some of the techniques [142] (sometimes used in conjunction with surrogate models) in general these are just random samples to lower the cost of optimization like in the case of the Hyperband algorithm [82]. The methods of sub-sampling training data will be revised in more detail in Section 2.3.2.

Grid search and random search

Grid search (GS) and random search (RS) are two popular methods for hyperparameter tuning [15]. Their main advantage is the simplicity of implementation (which makes them easily available in many software packages e.g., Scikit-learn [104]) and easy parallelization as each evaluation is independent of the others. GS works by evaluating the Cartesian product of a user-specified finite set of values. Assuming there are k parameters and each of them has n distinct values, it creates n^k configurations and trains a model for each of them. The best combination of hyperparameters values is then chosen based on the performance of the model on a validation set. It is easy to see that GS has an exponential time complexity of $O(n^k)$. In consequence, it is not well-suited for problems with large numbers of hyperparameters. What is more, even though it is considered an exhaustive search method, it is often not able to find global optimum in the case of continuous hyperparameters [15]. This has two main reasons: GS cannot exploit the well-performing regions by itself and requires a predefined finite set of values to test. Therefore GS should be considered only for relatively small configuration spaces (low dimensional spaces e.g., 2D) in order to be effective. There are many improvements already proposed to GS including:

- Automatic narrowing of search space, sometimes called hierarchical GS [128].
- Using subsamples of data for initial approximation of promising regions.
- Early stopping.

Due to its availability and simplicity it is still widely used for hyperparameter search in low-dimensional spaces.

RS is similar to GS in the manner of evaluating all of the candidates' configurations. The main difference comes in how these candidates are sampled in a random manner. The number of such samples is predefined and each hyperparameter is sampled based on specified distribution with lower and upper bounds. As the number of samples is fixed, there are only n unique configurations to be tested making its time complexity $O(n)$. In theory, if the configuration space is large enough, then the global optimum or its approximation can be detected. Another advantage over GS is that hyperparameters combinations coming from the specified distributions improves system efficiency by reducing the probability of wasting much time on a small poor performing region.

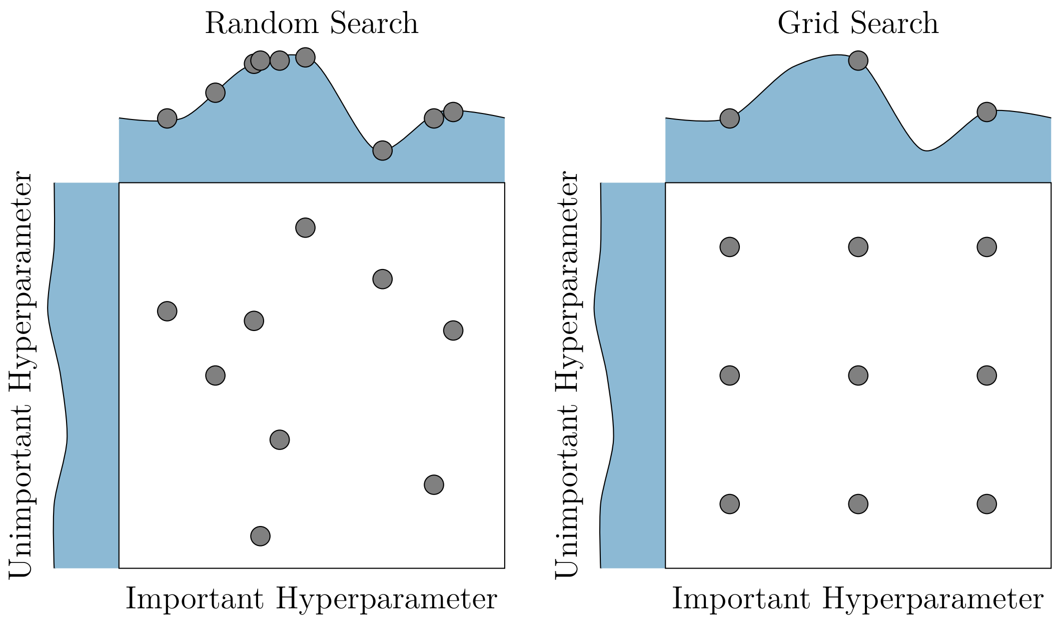


Figure 2.7: Example of RS and GS of nine trials for optimizing function $f(x, y) = g(x) + h(x)$. The importance of the hyperparameters is illustrative and can be obtained *a posteriori*. This figure is inspired by [15].

This situation is illustrated in Figure 2.7 where nine trials of different hyperparameters are presented. There are two different hyperparameters that differently affect end results, denoted as important and unimportant. In the case of GS points evenly cover the original 2D space while their projections over either of the hyperparameters produce an inefficient coverage of just three points. In contrast to that uniformly sampled points are less regular in the original space but far better cover each hyperparameter subspace. In that case, the advantage of RS is exploring nine different values of an important hyperparameter whereas GS checks only three. Although as discussed by Bergstra [15] this is a rule rather than an exception, GS could still be useful in selecting SVM hyperparameters as considering RBF kernel for example, there are just two hyperparameters to select and we limit the number of trials needed for grid search by sampling on a logarithmic scale between lower and upper bounds (as recommended by the LibSVM creators [24]). In both cases (RS and GS) their usage is much better than manually tuning as it saves time and increases the reproducibility of results (for RS it is important to get the same seeds for the random number generator).

There is still ongoing research in improving RS algorithms. The work by Florea et al. [45] focused on reducing the computational complexity of the RS method by limiting the number of trials needed to find a good solution. They achieved it by introducing a dynamically computed stopping criterion. That algorithm is divided into two parts. In the first one, a small number of predefined trials are evaluated and the best solution is remembered. In the second part, they try to find a new solution better than the one found before. If such a solution is found the algorithm is terminated. As presented by the authors, this led to a decrease in the number of RS evaluations (SVM trainings) needed without significantly lowering accuracy.

Other interesting works on improving RS are successive halving [72] and Hyperband [82]. The idea behind successive halving is to allocate budgets (B) for each of configuration (n). During the first iteration of the algorithm, all of the configurations are evaluated but within a limited budget ($b = B/n$). Then according to the evaluation results, half of the poor-performing hyperparameter configurations are eliminated and the rest is passed to the next iteration with doubled budgets. This process repeats until there is a final (the best) configuration found. The main difficulty in running such an algorithm is determining the trade-off between running a few configurations with a high budget or testing more configurations but with lower budgets.

Hyperband algorithm tends to solve this problem. It is a wrapper method of successive halving. It is considering multiple different values for the number of configurations (n_i) where each such number is allocated with its own budget. The larger the n_i is, the smaller budget is being assigned to it. Then for each such n_i the successive halving procedure is performed. Hyperband needs two parameters to be set: the maximum amount of resources R that can be assigned to one configuration, and the fraction η of configurations that are eliminated each time successive halving is run. The method starts with the most populous group for maximum exploration. Every following iteration reduces the number of configurations by a factor of η, until the final iteration where each configuration is given R resources, which is similar to a traditional random search. This approach allows Hyperband to take advantage of situations where adaptive assignments perform well, while still maintaining satisfactory results when conservative allocations are necessary.

Bayesian optimization

Bayesian optimization (BO) [116] is a powerful optimization technique that is particularly well-suited to problems where the number of function evaluations is limited. It determines the next hyperparameter value based on the previous results of tested hyperparameter values, which avoids many unnecessary evaluations. Thus, it can be considered a sample-efficient method. This approach is particularly useful when the objective function is expensive to evaluate, or when the function is a black box with no gradient information available. However, due to the fact that Bayesian optimization models operate based on the values that were previously tested, they are considered sequential methods and are challenging to parallelize.

In order to select the next values to test, BO builds a probabilistic model of the objective function (called a surrogate model) and then uses this model to guide the search for the optimal solution. The surrogate model aims to fit all the currently observed points into the objective function. Another important part of BO is an acquisition function which determines how new points will be sampled. It needs to balance the trade-off between exploration and exploitation. In exploration, the sample is used to gain knowledge about the areas that have not been sampled yet, while exploitations try to sample the currently most promising regions. Those promising regions are based on the posterior distribution. The general pseudocode for the Bayesian optimization algorithm is presented in Algorithm 2. This particular example uses the Gaussian process for a surrogate model.

Algorithm 2 Bayesian optimization algorithm with Gaussian process as a surrogate model. Kernel function refers to the covariance structure of a GP model.

```
1: Initialize an empty set of samples $S$
2: Select a kernel function and a prior distribution for the GP
3: for i = 1 to n do
4:     Select the next point $x_i$ to evaluate by maximizing the acquisition function (e.g., expected improvement)
5:     Evaluate the objective function $f(x_i)$
6:     Add $(x_i, f(x_i))$ to the set of samples $S$
7:     Update the GP model using $S$
8: return $x$ that maximizes $f(x)$
```

In general, Bayesian optimization methods are formalized as Sequential model-based optimization. Common surrogate models for BO include Gaussian Process (GP), Random Forest Regressions, and tree Parzen estimators [14]. This kind of optimization might seem like a perfect fit for any machine learning algorithm, as each evaluation of objective function is very expensive.

One of the recent examples of using Bayesian optimization with the GP is presented in predicting household vehicle ownership [137]. The authors used described above algorithm for BOGP. They were able to improve the results of the SVM model performance by several percentage points on the test set thanks to BO. What is more, the optimized SVMBO (as named by the authors) performed better than other tested machine learning models including k-nearest neighbors and decision trees.

A combination of selecting random subsets of data for initial evaluation and combining this process with BOGP was proposed by Klein et al. [77]. The authors propose a method called **FA**st **B**ayesian **O**ptimization on **LA**rge data**S**ets (FABOLAS) which, during sampling for the next solution, selects additional parameter s which is the fraction of data that will be used for evaluation ($s \in (0, 1]$)). While the goal is to optimize performance for $s = 1$, it is usually much cheaper to use smaller parts of the dataset while obtained values should still correlate across s domain. Based on some initial experimentation, the authors proposed to use a logarithmic scale for s as tested on MNIST dataset (a collection of 70,000 handwritten digits that is commonly used as a benchmark) for $s = 1/128$ already yielded representative results. This observation is also used in one of the algorithms presented in this work (see Section 3.5). FABOLAS is often able to discover efficient configurations at a much faster rate, typically 10 to 100 times faster, when compared to related techniques such as Multi-Task Bayesian optimization, Hyperband, and regular Bayesian optimization. The mentioned BOGP has its limitations as it is best suited for working with continuous variables. In the SVM case, this means that BOGP is used for optimizing hyperparameters of the kernel and C value (most often RBF with single γ hyperparameter is selected). Not being able to handle efficiently discrete, categorical, and conditional hyperparameters, other surrogate models were introduced. This includes Random Forest Regression [70] and Tree-structured Parzen estimator [14, 16].

Bayesian **O**ptimization **H**yper**B**and (BOHB) [42] is another cutting-edge method for tuning hyperparameters that combines the strengths of Bayesian optimization and Hyperband while avoiding their weaknesses. Traditional Hyperband relies on random search to explore the hyperparameter configuration space, which can be inefficient. BOHB addresses this issue by utilizing Bayesian optimization, which is more efficient and allows for parallel resources to be used to optimize all types of hyperparameters. BOHB uses TPE as its standard surrogate model for Bayesian optimization and employs multidimensional kernel density estimators. It has been demonstrated that BOHB surpasses many other optimization techniques when tuning SVMs and deep learning models. The only limitation of BOHB is that it requires that evaluations on subsets with small budgets are representative of evaluations on the entire training set (which is often true for SVMs [77]), if not it may have a slower convergence speed than standard Bayesian optimization models.

Building specialized kernels

Another approach to optimizing the SVM model is to build a new (custom) kernel function. This can include a combination of different kernel functions e.g., polynomial, linear, and RBF [26, 68]. Different valid kernel functions can be joined using their properties, so if a given function satisfies Mercer conditions and is considered a valid kernel the same is true about the sum or multiplication of two kernel functions. One more important thing is how some of the kernel functions e.g., RBF are considered local functions as they affect only some regions around SV (if the samples lay further from the SV the kernel function value is 0), while on the other hand linear and polynomial kernels are regarded as global functions as each support vector contributes to the final decision boundary. Most of the time joining those local and global functions tends to provide improvement in classification performance. While this approach can seem tempting, it often results in creating additional hyperparameters that need to be tuned.

Another possible approach could be the selection of the kernel from a set of predefined functions [4]. The so-called weak kernels can be used to find the best linear combination of those in order to apply it to SVM training. The experimentation shows promising results on the used datasets and robustness to problems where feature selection is important.

The technique of genetic programming has also been used for developing and refining kernels, including those specifically designed for natural language processing known as string kernels [120]. Additionally, kernel functions can be tailored and enhanced through the use of reinforcement learning [10]. The selection of the model can also extend beyond commonly-used kernels to include options such as Mahalanobis kernels [73] or kernels that are specifically suited for datasets with varying class densities, known as isolation kernels [127].

Metaheuristics

Last but not least metaheuristic methods were also used to tune SVM hyperparameters, these include, GA and particle swarm optimization as the two most prevalent metaheuristic algorithms. The GAs were already described in Section 2.2. Particle swarm optimization (PSO) is inspired by the behavior of a swarm of birds or a school of fish, by mimicking the behavior of a swarm of particles that move through the search space, where each particle represents a potential solution to the problem. The movement of particles is affected but the current best solution found (by any particle) and their own inertia and momentum parameters.

An example of applying such metaheuristics to optimize SVM can be found in work by Zhou et al. [145]. They employed three different algorithms: whale optimization algorithm, gray wolf optimization, and moth flame optimization for optimization of SVM hyperparameters (namely C and γ). The presented results were used to predict the advance rate of a tunnel boring machine which is a key parameter for tunneling projects. As presented the moth flame optimization provided the best results although the size of the population of 150 could be used only for small datasets.

Another example of a slightly modified version of PSO called switching-delayed-PSO is presented by Zeng et al. [141]. This modification delays the information of local and global best particles during the velocity update. It allows for improvement in the results by finding the current state of evolutionary computation (convergence state, exploitation state, exploration state and jumping-out state) and adjusting the parameters used by the heuristic. This allowed to provide state-of-art results for the diagnosis of Alzheimer's disease.

GAs were also successfully applied to the problem of SVM model selection [81]. The authors propose to combine multiple popular kernel functions and optimize their hyperparameters at the same time. They also used prior knowledge where the hyperparameters are sampled using a logarithmic scale. This early work presents interesting improvements over regular SVMs.

Another interesting extension to classical GAs is incorporating feature weighting into the evolutionary process [124]. It can serve a feature selection (when weight is equal to zero) purpose and further improve the quality of classification. This solution is then applied to intrusion detection problem on which it provides faster training time of SVM and decreased error rate.

Summary

As this problem is still very relevant not only for SVMs but also for other machine learning methods, there is a lot of research in this area. There are great review articles which can provide a good overlook of the field and current research problems as well as available methods [138]. However, it is hard to properly compare many of the presented techniques as they are often used in different sub-fields, where either datasets or implementations are not freely available or some specific knowledge during the processing could be used.

An interesting summary of currently available methods was made by Wainer et al. [128]. The authors focus purely on optimizing SVM hyperparameters with 18 different methods using over 100 datasets from the UCI repository. Their finding is that none of the presented algorithms shows significant improvement in terms of SVM accuracy in comparison to grid search algorithms. It is worth noticing that the authors took on a practical approach to using many advanced algorithms like PSO based on their default parameters available in the software packages, while e.g., population initialization could play a crucial role in many metaheuristic techniques. Although the performance of SVM is heavily dependent on hyperparameter selection all of the available methods perform equally well when comparing classification performance. It is important to mention that not all of the HPO algorithms take the same amount of time/evaluations (which is the main differentiation between those tested algorithms). Time performance could be a good selection criterion for which algorithm to choose. On the other hand, an argument to use methods that are easier available, easier to implement or tend to present other advantages (such as easy parallelization) could be made. Whichever method is selected, HPO will virtually always guarantee improvement of performance over some default values provided in software packages.

2.3.2 Selection of SVM training set

Generally, training models with enough information is essential to achieve good performance. However, it is common that a training dataset $\boldsymbol{T}$ contains samples that may be similar to each other (that is, redundant) or noisy. This increases the computation time and can be detrimental to generalization performance. The process of instance selection (also called training set selection) aims to select such subset $\boldsymbol{T'} \in \boldsymbol{T}$, which will represent the whole training and achieve similar performance while reducing necessary computation. The problem of the training set selection is similar to feature selection (described in 2.3.3). In the same way, these methods can be categorized in different manners:

1. **Incremental** method ($\boldsymbol{T'} = \emptyset$ at the start) or **decremental** method ($\boldsymbol{T'} = \boldsymbol{T}$), **Batch** (mark all instances that should be removed and remove them at

once), **Mixed** (preselect a subset, and repeatedly, add or remove instances from this subset), and **Fixed** (size of $\boldsymbol{T'}$ is predetermined) [48],

2. **Wrapper**, where the selection criterion is based on the accuracy obtained by a classifier or **Filter**, where the selection criterion uses a selection function which is not based on a classifier,
3. **Dependent** or **independent** of $\boldsymbol{T}$ cardinality in terms of computational complexity.

For this dissertation the categories presented in Figure 2.8 will be used to provide an analysis of the field.

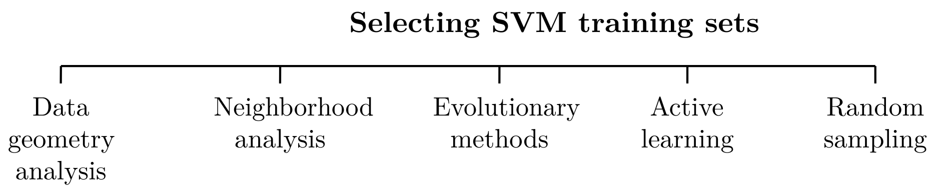


Figure 2.8: General categories of approaches for selecting SVM training sets. This figure is inspired by [98].

Commonly, SVMs can be successfully trained using just a subset of all the data that are available for training. There are RS approaches proposed. Similarly to HPO problems such methods are simple to implement and often sufficient for finding a reduced training set (at least if the size of the reduced set can be estimated *a priori*). Another of their advantage is not depending on the cardinality of $\boldsymbol{T}$. One of such approaches was proposed by Balcazar [11]. It works by starting with a random subset of $\boldsymbol{T}$ sampled according to the weights assigned to training vectors. Initially, an SVM classifier is trained using this random subset. Afterwards, the whole training set is analyzed to check which vectors are classified correctly. Then the weights for misclassified vectors are increased so they are more likely to be sampled in next iterations. If the number of iterations is large enough, the important vectors, including SVs, will have a higher weight than other vectors thus the refined set should be composed of those vectors. The main drawback of this algorithm is the need to set the desired size of $\boldsymbol{T'}$ beforehand. This might require many trials and could become time-consuming (especially for large datasets). Another problem is that random sampling may ignore any relations which occur in a dataset.

Data geometry analysis, on the other hand, is focused on exploiting such relations. These methods often select vectors from the regions lying close to the separating hyperplane. Of course, this would need to determine the approximate location of the hyperplane in order to find the most important vectors. A subgroup of data geometry analysis employs various clustering techniques to identify the clusters formed by the vectors from the same class [129, 33]. This approach makes it possible to eliminate vectors that are situated within these clusters, as they have a lower probability of being chosen as SVs during SVM training.

Shen et al. [114] proposed a method for eliminating unnecessary training set vectors by analyzing cluster boundaries and examining other inter-cluster relationships. The k-means clustering is used to obtain clusters, and for each cluster, a distance density set is calculated. Vectors that are close to the centroid are considered "dense", and those that are far from being considered "sparse". The authors use Fisher's discriminant analysis to find the boundary between the dense and sparse parts of each cluster and include only the sparse vectors for the selection of a refined set. Additionally, the authors consider removing redundant (one-class) clusters as there is a low probability of selecting SVs in such regions.

The main disadvantages of using clustering techniques are their time complexity and the need to analyze the entire training set. Additionally, there are different crucial hyperparameters of clustering methods which can further affect their results. Geometric information can also be obtained without using clustering. Methods like these include applying β-skeleton algorithms [144] or utilizing other basic classifiers [2] to get this information. Another example of such work includes the usage of the fast nearest neighbor condensation classification rule (FCNN) [7]. SVMs are combined with FCNN, where the vector selection criteria are based on the decision boundary. The FCNN process starts with an initial refined training set that is composed of the centroids generated for each class independently. For each vector, $a \in \boldsymbol{T'}$, a point belonging to the Voronoi cell (i.e., a set of $\boldsymbol{T}$ vectors that are positioned closer to a than any other vector in the current $\boldsymbol{T'}$) of a, but labeled with the opposite class, is included in the refined set. The algorithm continues until there are no more vectors from $\boldsymbol{T}$ to be added to $\boldsymbol{T'}$.

Another group of algorithms uses neighborhood analysis. It is assumed that the SVs are more likely to lie in heterogeneous regions of the dataset. This is likely to occur, as the decision boundary is often located close to such areas [58]. These algorithms include the usage of ensemble classifiers as done by Guo et al. [59]. They exploit a custom margin definition which is calculated for each vector based on the number of votes of the base classifiers for a given class. These margins lie in the range of $[0, 1]$ where smaller values indicate vectors closer to the hyperplane (as learned by the ensemble). The refined set for SVM is selected by sorting all of the vectors according to their margin values and selecting the smallest ones. As the base classifiers, the authors used decision trees and applied bagging to build an ensemble.

Another work proposes to exploit an induction tree to predict if the vector is likely to become a SV after SVM training [22]. To construct the tree, an SVM is initially trained using a small randomly selected subset of $\boldsymbol{T}$ to identify which vectors are selected as SVs. The selection process in this method employs a simple heuristic to balance the $\boldsymbol{T'}$ for SVM training. Afterwards, vectors are passed to the tree to indicate the potential best candidates for SVs. These are included in $\boldsymbol{T'}$. In summary, in order to effectively use methods that analyze the data geometry or local neighborhood, it is still necessary to examine the entire $\boldsymbol{T}$ to choose the valuable subset.

Evolutionary algorithms, such as genetic algorithms, have been shown to be effective in finding a reduced training set for SVMs. They typically use a wrapper approach, where the fitness of a reduced set is determined by training an SVM with that set. The population of reduced sets is then evolved to find the best one [75]. Similarly as in the case of random selection [11] it can be difficult to select the optimal size of a reduced set. This led to the development of more advanced adaptive strategies which can tune the size of $\boldsymbol{T'}$ during the optimization process [95]. The adaptation is based on observing the performance of the whole population and looking for certain signals (e.g., number of SVs in relation to the size of $\boldsymbol{T'}$). Depending on their occurrence, the size of the reduced set can be increased. It is important to note that such a method usually starts with a very small refined set. More advanced methods incorporating additional information collected during the evolution were also proposed [97, 96]. These specialized evolutionary methods (memetic algorithms) incorporate additional local search procedures in order to better balance exploration and exploitation of the search space.

Another interesting work proposes to incorporate multi-objective optimization techniques as the goal of the training set selection is the reduction of the size of the original $\boldsymbol{T}$ while maximizing the classification performance (e.g., accuracy). Cheng et al. proposed a method called SDMOEA-TSS [28], which uses multi-objective optimization techniques to reduce the size of the training set while maximizing classification accuracy. The method divides the objective space into several subregions to create a diverse population. The algorithm can be adjusted to prioritize either reducing the size of the training set (SDMOEA-TSS(A)), increasing accuracy (SDMOEA-TSS(C)) or finding a balance between the two (SDMOEA-TSS(B)).

One of the key advantages of the evolutionary methods is that they are independent of $\boldsymbol{T}$ cardinality when selecting reduced sets. Although these methods provide state-of-the-art results, they often require the initial optimization of the SVM model, such as determining the appropriate kernel function and its hyperparameters, which is usually done through a time-consuming grid search [28, 96]. In the next section, the **M**emetic **A**lgorithm to Select Training Data for **SVM** (MASVM) [96] will be presented. This is one of the evolutionary approaches for the training set selection designed specifically for SVM and employing the knowledge about SVM classifier. This section is important as it provides insights into the implementation of MASVM and describes this algorithm in detail.

Memetic Algorithm to Select Training Data for Support Vector Machines (MASVM) [96]

In this subsection, the MASVM algorithm will be discussed in more detail as it is utilized for the training set optimization among proposed algorithms. The overview of this algorithm is presented in Figure 2.9.

It starts by creating a population of N individuals by randomly sampling the training set. Each chromosome is built of K_t vectors from each class (only binary problems are considered so the final size of the chromosome is $2 \cdot K_t$). Thus each

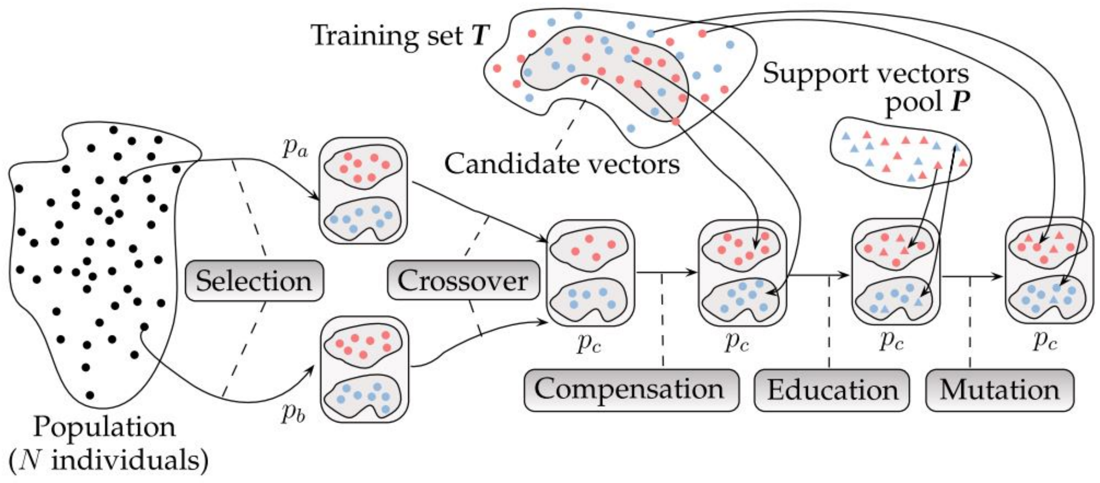


Figure 2.9: Overview of MASVM algorithm. Source: [98].

chromosome represents a refined training set $\boldsymbol{T'}$. Once the population is created, the fitness of all individuals is calculated. The fitness calculation includes training N distinct SVMs using the $\boldsymbol{T'}$ from chromosomes and evaluating their performance on the validation set.

After this initialization phase, N pairs (p_a, p_b) of individuals are selected using the local-global adaptation scheme (LGA), where $p_a \neq p_b$. In this approach, the population is arranged in descending order based on their fitness values and divided into two groups. The first group includes the top $(\epsilon \cdot N)$ individuals with the best fitness, while the other group includes the remaining $(N - \epsilon \cdot N)$ individuals. In the local selection mode, $\epsilon \cdot N$ pairs are created from the well-fitted (first) group while the rest are randomly drawn from the less-fitted part (N pairs are determined to create new population with the same size as initial one). In the global mode, pairs are created by selecting one individual from each group to promote balance and exploration of the solution space. The selection method is adaptively changed during the algorithm's execution.

After creating N pairs, a single child is generated for each pair. This new individual inherits all of the vectors from both of the parents ($p_c \leftarrow p_a \cup p_b$) while maintaining their uniqueness. The size of the resulting reduced training set $|p_c|$ is randomized, and equals $\max(|p_a|, |p_b|) \leq |p_c| \leq 2K_t$. If the number of distinct vectors inherited from p_a and p_b is smaller than selected $|p_c|$, additional vector are randomly drawn from $\boldsymbol{T}$ (compensation operation),

In the next step, the education process is employed. There is a chance (determined by the P_e) that training vectors in the offspring solutions that were not selected as SVs (in previous SVM training) will be replaced with random vectors selected from the pool of support vectors. This pool contains all vectors that were chosen as support vectors during the evolution process. Afterwards, the newly created population (P') undergoes mutation with a probability m_m. In this step, a certain percentage (f_t) of vectors in each individual are selected and replaced with random vectors from the $\boldsymbol{T}$.

To further take advantage of the vectors that were determined to be important during the evolution process (as they were selected as SVs), the so-called super individuals (SIs) are created and evaluated. The number of SIs is equal to $\alpha \cdot N$, where $1 \geq \alpha \geq 0$. These super individuals consist of up to $2K_t$ randomly chosen and distinct support vectors from the support vector pool. Up to K_t vectors are picked for each class. Note that in case there are not enough vectors in the support vector pool ($\mathcal{S}_{\text{pool}}$) this number could be lower.

In the selection of individuals for the next generation the size of the population is kept constant and the best N individuals are chosen from the previous population (P), current (intermediate) population (P'), and the set of SIs (P^{SI}).

In the end, the hyperparameters of the algorithm are adapted and set $K_t = \rho_K \cdot K_t$, where $\rho_K = 1 + (\varrho_{sv} - \mathcal{T}_{\text{SV}})/(1 - \mathcal{T}_{\text{SV}})$, $\mathcal{T}_{\text{SV}}$ is the SV threshold and $\varrho_{sv} = \text{SV}(P'_{\text{best}})/(|P'_{\text{best}}|)$, $\text{SV}(P'_{\text{best}})$ denotes the number of SVs in the best individual from the current population, and $|P'_{\text{best}}|$ the size of P'_{best}, alongside. The K_t value can be increased only once in $\mathcal{T}_s$ generations. This help to fully exploit the current value of K_t. Additionally, K_t will remain constant if the average fitness in the population did not increase since the last generation ($\Delta\eta = 0$), or the improvement was greater than $0.5 \cdot \Delta\eta$. If in the last $\mathcal{T}_s$ generations K_t was not updated and the current LGA selection scheme is in the local mode it is switched to the global mode. This provides a better exploration of the solution space. Once K_t is updated, the LGA is set to local mode again.

The evolution continues until the termination condition has been met. The original proposed stop condition included multiple possible conditions: fitness not growing in a given number of consecutive generations, K_t is not further increased or the desired fitness is reached. During the mentioning of MASVM in this dissertation the average population's fitness is monitored—if its improvement is smaller than $\eta_{\min}$ in two consecutive generations the algorithm is stopped. This condition is based on the observation that the population's diversity is likely very low (as average fitness did not change), thus the probability of improving the fitness of the best individual drastically decreases [119].

One important difference in the above description compared to the original MASVM [96] is omitting of the regeneration process as suggested by Nalepa and Kawulok in subsequent work [97]. This should decrease the running time of the algorithm while it should not affect the end results in terms of the quality of selected $\boldsymbol{T'}$.

2.3.3 Feature selection

Feature selection is the process of identifying a subset of the most relevant features from a larger set of features for a given prediction task. The goal of this process is to improve the performance of the model by reducing the dimensionality of the input space, and by eliminating irrelevant, redundant, or noisy features that can negatively impact the performance of the model. There was a tremendous effort to develop different methods to address this problem. Feature selection methods can be broadly classified into three categories [13]:

1. **Filter methods**—these methods evaluate the relevance of features based on statistical measures or domain knowledge, and select a subset of features independently of the classification model. Examples of filter methods include correlation-based feature selection, mutual information, and chi-squared test [122, 117].

2. **Wrapper methods**—these methods evaluate the relevance of features by training and testing different models with different subsets of features, and selecting the subset that optimizes the performance of the model. Examples of wrapper methods include recursive feature elimination [6], GAs [86], and simulated annealing [1].

3. **Embedded methods**—these methods incorporate feature selection into the model training process, by adding a penalty term or a constraint to the optimization function that encourages sparse solutions. Examples of embedded methods include Lasso regression, Ridge regression, and Elastic Net [23, 102].

In practice, the choice of feature selection algorithm depends on the nature of the data, the size of the feature space, computational resources, and the desired performance metrics. For example, filter methods are often used for high-dimensional datasets, while wrapper and embedded methods are better suited for smaller feature spaces [18].

One interesting example of the filter method is proposed by Yao et al. [139]. A new ensemble feature selection method called FS-MRI, which integrates multiple ranking information, has been introduced. The FS-MRI method can automatically determine the threshold function based on the model performance, resulting in the selection of an optimal and stable subset of features. The approach incorporates nine different feature selection methods into its framework.

The SVM Recursive Feature Elimination (SVM-RFE) [60] is a wrapper-based feature selection approach that utilizes the SVM as the base classifier. The method uses the objective function $(1/2)w^2$ (see Equation 2.8) to rank the features by their discriminatory ability. At each step, the feature with the smallest ranking score is eliminated, based on the corresponding component of the weight vector. This process is recursively repeated until all features are ranked in order of importance, using a backward feature-elimination scheme that excludes insignificant features.

A more recent work propose to couple SVM RFE with Binary Biogeography Optimization (BBO) [6]. In order to improve the quality of solutions obtained through the mutation operator and strike a balance between exploitation and exploration, SVM-RFE is embedded into BBO. The resulting approach, BBO-SVM-RFE, shows promising potential in effectively searching the feature space to obtain the optimal combination of features. The obtained results indicate that BBO-SVM-RFE is a reliable method for feature selection.

Besides that, other works in this area include the wrapper method of feature selection with a lot of work put into using evolutionary algorithms [52, 53]. In general, each individual in EA is coding a separate set of features. After applying an appropriate method to produce new solutions the SVM is trained using those feature sets and evaluated using the selected metric. Neumann et al. [101] proposed the modification in the SVM training process to gain information about features that should be used. In order to simultaneously select features and construct a model, the authors have introduced an additional term to the standard cost function of SVM. This added term penalizes the cardinality of the selected feature subset, allowing for the optimization of a modified cost function.

The choice of feature selection algorithm depends on the data, the size of the feature space, computational resources, and the desired performance metrics. With the increasing availability of large datasets, efficient and effective feature selection methods are becoming more important for building accurate and efficient machine learning models.

2.3.4 Multi-aspect SVM optimization

The issues of selecting the appropriate training samples and features, as well as optimizing the hyperparameters of SVMs are closely related, thus there are many methods to address these problems together. The problem where multiple optimization techniques are involved will be referred to as multi-aspect optimization.

Raman et al. proposed a new method for optimizing the hyperparameters and features in SVMs by introducing a hypergraph GA [109]. They included the features selection in the chromosome encoding, along with γ and C values of binary classifiers, which were used to solve multiclass classification. Another similar approach was used in another work, which utilized a nature-inspired multi-verse optimizer [43]. Several studies have shown the effectiveness of combining model selection with feature selection [125, 71, 5]. PSO was used to achieve this integration in the work by Huang et al. [66]. These methods demonstrate the usefulness of simultaneously selecting the most valuable features and optimizing the model's parameters.

SVM kernel evolution may also be combined with training set selection. In research by Nalepa et al. [99] a scheme in which the kernels are elaborated using a neuro-fuzzy system and the training set is selected afterward using an evolutional algorithm.

There were also several attempts for simultaneous selection of training sets and features. Garcia-Pedrajas et al. proposed a memetic technique [50] with four different local search procedures and a new fitness function which provides a proper balance between selecting the training samples and removing the features. Such combined selection can also be performed in a static manner, to identify the inactive features and samples before the SVM optimization [143]. The article [83] proposes a hybrid memetic algorithm that combines variable neighborhood search and a memetic algorithm to simultaneously select instances and features. The proposed method for feature and instance selection is utilized to reduce the size of data and train a predictive model for later performance evaluation. Compared to other metaheuristics, the proposed approach strikes a balance between exploration and exploitation, and the results demonstrate its superior robustness over other feature selection techniques.

In the context of SVM model optimization, simultaneous optimization of the three interconnected elements—features, training samples and model—has been shown to lead to improved performance and classification scores. Previous works have focused on optimizing two out of the three elements. However, integrating feature selection into the evolutionary process and optimizing all three elements together in an alternating or simultaneous manner, as demonstrated in the works co-authored by the author of this dissertation [39, 37, 38], has resulted in models with a lower number of SVs and features while maintaining high classification accuracy.

2.3.5 Building classification ensembles

Ensemble learning is a technique used in machine learning to improve the performance of a model by combining the predictions of multiple base models. The idea behind ensemble learning is that multiple models, each trained on the same dataset (not necessarily on the same subset of data), can provide different perspectives on the problem, and by combining their predictions, the final model can be more robust and accurate than any of the individual models. There are several popular methods for building ensembles in machine learning, including:

1. **Bagging**: This method involves training multiple models on different subsets of the training data, and averaging their predictions. This can be used to reduce the variance of the model and can be applied to both regression and classification problems.

2. **Boosting**: This method involves training multiple models in sequence, where each model is trained to correct the mistakes of the previous model. This can be used to reduce the bias of the model and is most commonly used for classification problems.

3. **Stacking**: This method involves training multiple models on the same data, and using their predictions as input to a meta-model that makes the final prediction. This can be used to improve the performance of the model by combining the strengths of different models.

Ensemble methods generally require more computational resources than separate models, but they can provide significant improvements in model performance. They are especially useful when working with complex problems or large datasets.

There are multiple works that tried to build ensembles of SVMs. Early work proposed a cascade of SVM [57] that can efficiently parallelize and handle very large problems with hundreds of thousands of training vectors. The training data are divided into subsets and on each part, a separate SVM model is trained. The partial results are combined and filtered in a cascade of SVMs until the global optimum is reached. This approach is memory-efficient and faster than a regular SVM, even with a single pass.

A similar strategy was used by Claesen et al. [31] where multiple SVMs are trained separately on different subsamples of the $\boldsymbol{T}$, ultimately leading to faster training. To further increase the efficiency, the SVs are analyzed and shared among models so if the same vector is selected as a SV in multiple models it needs only one calculation of the kernel function. This, however, requires the base model to share a single kernel function with the same hyperparameters. One more problem with that approach is that the size of subsamples used for training needs to be set beforehand so it becomes another hyperparameter that has to be tuned.

The GenBoost-SVM method [110] uses an adaptive boosting algorithm. The authors examined different pre-selections of a training set using GAs to reduce training times and tackle imbalanced data. Additionally, diversity and early stopping were considered to reduce the generalization error. Using different SVM kernels was shown to improve the performance of such a classifier. It can be already noticed that all of the presented works used some methods to reduce the size of the training set. Pławiak et al. [107] presented a deep genetic cascade ensemble of classifiers. This work presented a stacking classifier with 16 layers of SVMs that are optimized using a GA. Although the presented results show great promises in credit risk scoring achieving state-of-the-art performance, the study focused on small datasets. It might be difficult to scale this solution to big datasets due to the high computational cost.

The already mentioned systems use SVMs only as the base models which makes them a homogeneous ensemble. Joining multiple different classifiers could also boost the performance, as the learning algorithms tend to produce different decision boundaries. That heterogeneity is crucial in an ensemble setting. One such example is joining SVM prediction with k-nearest neighbors algorithm [136]. Using stacking for building ensembles with three different base models [100] was tested by Nanglia et al. It was proved to provide increased performance in comparison to homogenous ensembles as well as other machine learning models.

2.3.6 Summary

The field of machine learning is constantly growing, with a vast number of solutions available for each problem, such as hyperparameter optimization, training set selection, and feature selection. While various techniques have been used in a joint setting, there is still a lack of a single method that can cover all these areas.

The solution proposed in this dissertation is introducing a new method that can efficiently cover hyperparameter optimization, training set selection, and feature selection simultaneously. Moreover, this method could be extended to build a new kind of ensemble that will utilize both SVM base models and random forest models to provide a heterogeneous ensemble. Such an ensemble could improve the performance of SVMs, and leverage the advantages of simultaneous optimizations. By combining these different techniques into one cohesive framework, one can leverage their strengths to build more robust and efficient machine learning models.

Chapter 3

Proposed methods

In this chapter, all of the newly introduced methods are presented. The chapter is structured around describing the building blocks of the methods and putting them into the context of how they were designed and used. Those techniques focus more on combining different aspects (such as training set selection and hyperparameter optimization) rather than on improving a single one of them individually. To make it easier for the reader to clearly see which algorithms were used in a given method, they are summarized in Table 3.1. Methods that are seen in the left part of the table are presented in a chronological order (the same as described in the sections). Some of those methods (e.g., ALMA, SE-SVM, or CE-SVM) can be treated as a general framework for the optimization of SVM so they re-use the same algorithms for hyperparameter optimization or training set selection. In the early work, it was presented that these solutions could be easily replaced, where GASVM [75] in ALGA [76] method was changed to MASVM [96] creating ALMA method while the alternating optimization schema remained unchanged. To not double the information the common building blocks of the methods are described separately. The chapter starts with a description of the genetic algorithm for hyperparameter optimization (GAHP) in Section 3.1 followed by the evolutionary algorithm for feature selection (EFS) in Section 3.2. Moreover, these algorithms (GAHP and EFS) are never used separately (as methods of their own), meaning that neither of them is tested later in experimental validation (Section 4.2). One more of the important building blocks of those algorithms is the training set selection. This is done by MASVM, proposed by Nalepa and Kawulok [96], and discussed in detail in Section 2.3.2. As this is an algorithm proposed in literature it is briefly tested to compare it with the proposed methods.

Table 3.1: A summary of algorithms used in the proposed methods.

Method	Section	Algorithm used				Simultaneous optimization
		GAHP (Section 3.1)	MASVM (Section 2.3.2)	EFS (Section 3.2)	RFE (Section A)	
ALMA [39]	3.3	✓	✓	✗	✗	✗
SE-SVM [38]	3.4	✓	✓	✓	✗	✓
ARBF-SVM [94]	3.5	✗*	✓*	✗	✓	✓
DA-SVM [40]	3.5	✗*	✓*	✗	✓	✓
CE-SVM [41]	3.6	✓	✓	✗	✗	✓
ECE-SVM	3.6	✓	✓	✗	✗	✓

* ARBF-SVM and DASVM are not using GAHP but they still optimize SVM hyperparameters.
The MASVM algorithm is modified in those methods as described in Section 3.5

The rest of this chapter is structured as follows. First, the method for alternating optimizations is presented in Section 3.3. Afterwards, the simultaneous optimization is described (Section 3.4) followed by a specialized algorithm that creates an adaptive kernel in Section 3.5. Finally, the algorithms for building an ensemble classifier in form of the cascade are presented (Section 3.6). The chapter is finished with a summary of the methods in Section 3.7 that also briefly discusses works not described here in detail.

During this chapter there is a lot of symbols are introduced for the description of the algorithms, all of those are gathered in Table 3.2 where first the general symbols applicable in multiple methods are presented and later section of the table presents symbols used only in specified algorithms.

3.1 A genetic algorithm for optimizing SVM hyperparameters

As discussed in Section 2.3.1 and presented in Section 2.1.3, the performance of SVM classifier is heavily dependent on its hyperparameters. One of the popular techniques to perform this optimization is GS algorithm [96, 75]. However, GS requires multiple training of SVM classifier which is difficult for large datasets. Whereas, the methods for training set selection were already introduced to tackle this

Table 3.2: Summary of symbols used for algorithms description.

	Symbol	Description
General	C	Hyperparameter of SVM, penalty for misclassifying training examples
	γ	RBF kernel hyperparameter
	$\boldsymbol{T}$	Training set
	$\boldsymbol{V}$	Validation set
	Ψ	Test set
	η	Fitness value
	$\mathcal{M}$	Set of SVM hyperparameters
	$\boldsymbol{T'}$	Reduced training set, $\boldsymbol{T'} \in \boldsymbol{T}$
	N	Size of the population
	P'	New population created after crossover operator
	P'_{SI}	Population of super individuals
	p_a	Chromosome/individual in the population
	$\boldsymbol{F}$	Features set
	$\boldsymbol{F'}$	Reduced set of features, $\boldsymbol{F'} \in \boldsymbol{F}$
	$\|\boldsymbol{T}\|$	Size of the training set
	c	Number of classes/lables
	K_t	The number of class examples selected for $\boldsymbol{T'}$
GAHP	α	Parameter of crossover operator
	m_m	Probability of mutation
	u	Mutation scale
	β	Upper range for mutation scale
	l	Epsilon for fitness improvement in stop condition
EFS	K_f	Size of the selected feature set
	F_a	Feature set of individual
	σ	Threshold for selecting features based on variance
	τ	Threshold for number of features after preprocessing
	λ	Threshold for how many individuals are considered during feature pool update
	E_p	Education probability
	E_r	Education replacement parameter
	m_f	Mutation probability
	m_r	Mutation replacement parameter
	$\mathcal{F}_{\text{pool}}$	Pool of features used for education
ALMA	Q	Population of $\mathcal{M}$
	M	The size of $\mathcal{M}$ population
	q_{init}, q_i	Individual/chromosome from $\mathcal{M}$ population
SE-SVM	K_t	Hyperparameter for size training set
	P^m	Part of population with encoded $\mathcal{M}$
	P^t	Part of population with encoded training set
	P^f	Part of population with encoded feature set
	$\mathcal{S}_{\text{pool}}$	Pool of support vectors build during evolution
	P_{best}	Population of best solution of saved before each regeneration
	R	Number of regenerations
DA-SVM/ARBF-SVM	$\|\boldsymbol{T}_{\text{min}}\|$	The number of training vectors in the least numerous class
	$\vec{\gamma}$	Vector of γ values
	$\mathcal{S}_{\text{best}}$	Set of previously selected SVs with their γ's values
	$\vec{\gamma}_\eta$	Vector with fitness of best-fitted individual for each γ
	G	Set of all populations in co-evolution scheme
	P^γ, P^i	Indication of certain population from G
CE-SVM	$\boldsymbol{T}_u$	Uncertain part of training set
	$\boldsymbol{V}_u$	Uncertain part of validation set
	$\mathcal{H}_{margin}$	Certainty threshold hyperparameter
	r_T	Indication if the size of training set is reduced
	r_V	Indication if the size of validation set is reduced
ECE-SVM	ζ	Hyperparameter for sampling rate of validation set
	ϑ	Hyperparameter of maximal size of validation set
	Υ	Hyperparameter of decreased of uncertain size
	ε_i	New cascade built in ECE-SVM
	ς_i	Node-wise MCC scores for new cascade

issue, in a majority of cases they used GS for SVM hyperparameters optimization. The goal of introducing the GA for SVM hyperparameter optimization is to improve on the existing training set selection method which uses evolutionary computation. The motivation to go for a GA is that it should be better than GS or RS techniques while it can still be partially parallelized (during the evaluation of individuals). Moreover, it will be natural to join it later with training set selection into simultaneous optimization process (using EAs). On the other hand as presented in [128] even quite simple method should be sufficient to improve upon currently available solution.

In the initial work, the focus was put on the usage of RBF kernel as it is the most popular one. There are multiple reasons why this type of kernel is a popular choice nowadays. Some of those came from its mathematical properties. It is a stationary kernel, which means that it is invariant to translation. A stationary kernel will yield the same value $K(\mathbf{x}, \mathbf{y})$ for $K(\mathbf{x} + \mathbf{c}, \mathbf{y} + \mathbf{c})$, where $\mathbf{c}$ is a translation vector whose dimension matches the inputs (the linear kernel does not have such a property). The single-parameter version of the RBF kernel has the property that it is isotropic, i.e., the scaling by γ occurs the same amount in all directions. Moreover, it maps vectors into infinitely-dimensional feature space [62]. On the other hand, it was already tested and found to yield the best results in multiple different applications such as text classification [56], hyperspectral image analysis [78], cancer genomics [67] or medical imaging [106]. What is more, throughout the years some intuition behind the γ hyperparameter grew where it can be thought of as a range of influence of SVs. All of that contributed to first selecting RBF kernel and trying to further improve the SVM performance with it.

The coding of chromosomes is one of the most important steps in designing a GA. For the optimization of hyperparameters that are represented as real numbers, it is natural to use them for representation of the problem (real-coded GA)[61]. In the presented solution considering the RBF kernel, there are two hyperparameters C and γ, however, this algorithm can be used to optimize any kernel (as it is later shown in Section 3.6 when multiple populations are evolved independently).

The outline of the algorithm is presented in Figure 3.1. On the right-hand side of this diagram, the process of fitness (η) evaluation is presented. In order to calculate the fitness of an individual (p_i), first the SVM is trained using the $\boldsymbol{T}$.

Afterwards, the whole validation set ($\boldsymbol{V}$) is classified and the confusion matrix is calculated. Based on the chosen metric, the final fitness value for the chromosome is assigned.

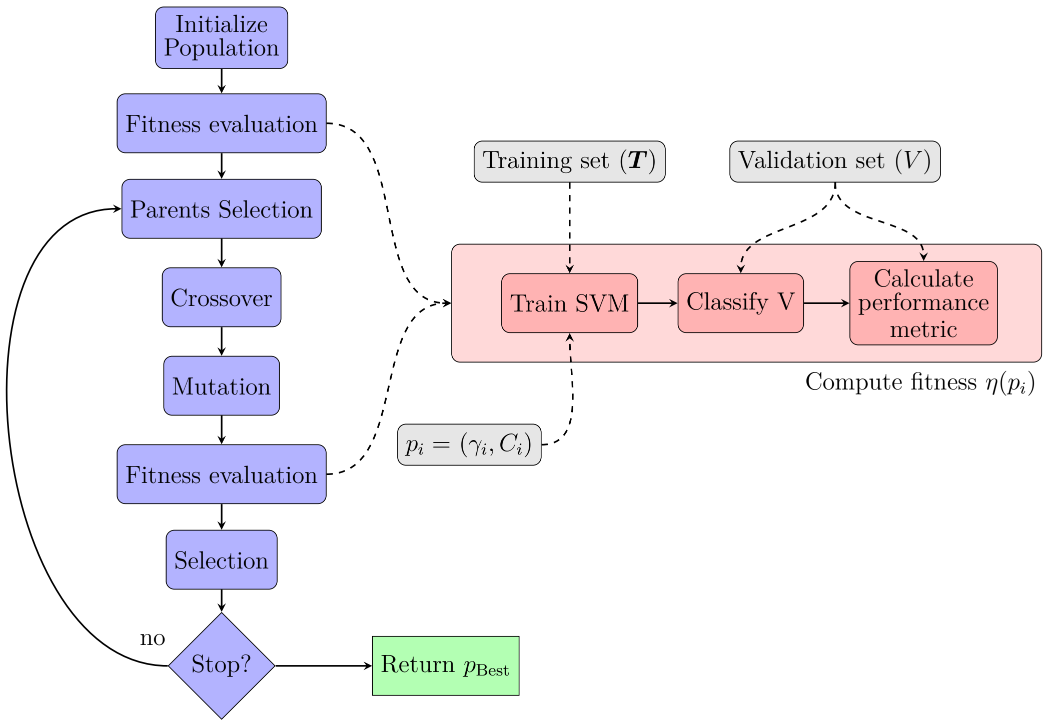


Figure 3.1: A flow diagram of the GA used for hyperparameter optimization.

The algorithm starts by creating the initial population. It is created to cover the initial setup range of values for hyperparameters in a logarithmic way. It calculates the combinations of hyperparameters based on the population size to create a grid. When the size of the population cannot be divided into grid, one individual gets $\gamma = 1/|\boldsymbol{F}| \cdot var(\boldsymbol{T})$, where $var(\cdot)$ is variance of the dataset, and $|\boldsymbol{F}|$ is number of features, $C = 1$ (this heuristic is based on Scikit-learn library [104]). The rest of individuals is sampled in a random manner in range specified in the algorithm configuration. This initial population is then evaluated and N parent pairs are selected for the crossover process (where $p_a \neq p_b$, p_a and p_b are chromosomes). It uses global-local selection described in Section 2.3.2 using always the global mode in order to provide a better exploration of search space and eliminate preliminary convergence. This means that the population is always divided into two parts of equal size and from each part one chromosome is selected in a random manner.

The crossover operator creates a new chromosome (p_c). In total, N chromosomes are created and a new population (P') is created (that has the same size the initial one). A new individual is created by using a heuristic crossover [61]. For each hyperparameter, the following equation is applied:

$$p_c = p_b + \alpha \cdot (p_a - p_b), \tag{3.1}$$

where α is a random value drawn from $[0.5, 1.5]$, p_a is a parent with higher fitness value compared to parent p_b. Although α was also proposed to be a constant value, the random one should bring more diverse children, hence enhancing the diversity of the population (even if the same pair is selected twice it can still produce different offspring). For $\alpha = 1$ the offspring will become the same as the parent p_a while for a value equal to zero it will become a copy of the parent p_b. Since GAs should pass on better genes to the next generation the range of random values should be symmetric with respect to 1. In this way, new individuals will be allowed to be in the space around the better of the selected parents.

Afterwards, each individual goes through the mutation process with a probability of m_m. This operator performs a form of a local search. If the individual is selected for mutation for each of its values the following equation is applied:

$$p_c = p_c + (p_c \cdot u \cdot \rho), \tag{3.2}$$

where ρ is a random sign with equal probability and u is a random percentage value drawn from range of 0 to β. The value of β needs to be relatively low and express the maximal percentage value of change. After applying all of the genetic operator's population of size $2 \cdot N$ goes through the selection process. This process selects the N fittest individuals to keep the population size constant. The stop condition is based on the growth of the average fitness of the population in two subsequent generations. If this growth is smaller than the set threshold (l), there is a little chance that the solution could be further improved and the algorithm ends.

3.2 Evolutionary algorithm for feature selection

As argued in Lessmann et al. [81], this process could be viewed as a preprocessing step where possibly expert knowledge about the problem at hand could be used. On the other hand, feature selection is a complicated problem and is mutually dependent with both model optimization as well as the training set selection. Both of those approaches were tested, where preprocessing is done by known in literature recursive feature elimination with cross-validation (RFECV) [6] (described in Appendix A). In the following section, an evolutionary algorithm for feature selection will be presented. This algorithm is used in the SE-SVM method 3.4.

Algorithm 3 Pseudocode for evolutionary algorithm for feature selection.

```
1: $\mathcal{F}_{\text{pool}} \leftarrow \emptyset$
2: FeatureProbability $\leftarrow$ Preprocessing($\boldsymbol{T}$)
3: $P \leftarrow$ RoulletteWheelInitialization($N$, FeatureProbability)
4: for all $p_i \in P$ do
5:     $\eta_i \leftarrow$ CalculateFitness($p_i$)
6: repeat
7:     $\mathcal{F}_{\text{pool}} \leftarrow$ UpdatePool($P$)
8:     Parents $\leftarrow$ SelectParents($P$)
9:     $P' \leftarrow$ Crossover(Parents)
10:    Education($P'$, $\mathcal{F}_{\text{pool}}$)
11:    Mutation($P'$)
12:    $P'_{SI} \leftarrow$ SuperIndividualsCreation($\mathcal{F}_{\text{pool}}$)
13:    for all $p_i \in (P' \cup P'_{SI})$ do
14:        $\eta_i \leftarrow$ CalculateFitness($p_i$)
15:    $P \leftarrow$ Selection($P \cup P'_{SI} \cup P'$)
16:    Adaptation($P$)
17: until StopCondition
18: return Best($Population$)
```

This algorithm was inspired by the MASVM (Section 2.3.2). Feature selection could be seen as a problem that is similar to instance selection in some manner. It aims to find the reduced set of features to improve the performance of the classifier and get rid of noisy features. The instance selection (training set selection) has similar aims, the main difference comes from the reduction of computational complexity, where often it could become more important, for instance selection, to reduce training set size drastically in order to apply computationally complex methods. In the case of feature selection, it might not always be true (please note that this work considers datasets with $|\boldsymbol{T}| \gg |\boldsymbol{F}|$). Based on this observation, the same coding of chromosomes as in MASVM (see Section 2.3.2) will be used for the feature selection process.

The outline of the method is presented in Algorithm 3. In order to initialize the population, the algorithm starts with preprocessing of datasets and ranks $\boldsymbol{F}$. The preprocessing set (line 2) consists of running four distinctive algorithms: mutual information, variance thresholding, RFECV, and stability selection. The RFECV utilizes ExtraTree (ET) [51] as a base model and returns a set of selected features which is converted into equal probabilities. So the whole vector of features' importance sums to 1. RFECV is presented in detail in appendix A. Variance thresholding computes the ANOVA F-value providing scores for each feature. The σ is then selected which means that only the top σ of features are selected and the rest gets a score equal to zero. Those scores are then rescaled to sum 1 (so they can be treated as probabilities). Mutual information is calculated for each feature and target variable (the class to which the vector belongs). These scores always have positive values, and similar to previous methods, are normalized to sum to 1. Finally, the stability selection algorithm [87] uses logistic regression with L1 penalty to estimate feature importance. The values obtained by each of the algorithms are then averaged. These scores are used as probabilities in roulette wheel initialization of feature sets, where K_f features are selected (in line 3). On initialization, only the top τ of features are considered for getting into the chromosome. This should help to provide better individuals at the beginning which is crucial for running simultaneous evolution of training and feature set, and SVM hyperparameters ($\mathcal{M}$) (see Section 3.4)

Afterwards, the fitness of all chromosomes in the population is assessed (line 5) by training SVM classifiers and calculating the confusion matrix on $\boldsymbol{V}$. The N pairs are selected for crossover using the local-global adaptation scheme (the same as in MASVM algorithm). The population is divided with ϵ into two parts (based on fitness values). The algorithm starts in global mode meaning that parents in each pair come from different parts of population.

Then, for each pair, a crossover operator is applied. It works by summing up feature sets ($F_c \leftarrow F_a \cup F_b$). As a summation of sets may lead to uncontrolled growth in the size of features selected up to K_f are drawn randomly. Although initially all of the chromosomes have the same size of K_f in later stages of algorithms this changes. As K_f is adapted dynamically, there could be individuals that differ in the size of the feature set. All of those newly created chromosomes create a new population (P'). After P' is created a memetic operation of education is performed. It provides better exploitation of information about the previous individual to enhance the current solution. This information comes from a pool of features that are associated with high-ranked individuals. The pool of features is created and updated (in line 7) by taking λ of the best individual in the population and creating a histogram of features (counting how many times a given feature is present in the population). Only features that occur more than once in the histogram are left in $\mathcal{F}_{\text{pool}}$. Once the pool is created, education is performed with a probability of E_p and replaces E_r of features to the ones that are in $\mathcal{F}_{\text{pool}}$.

Next, each individual is mutated with a probability of m_f. Mutation works by replacing m_r features with ones that are above mean in ranking used for initialization. The initial ranking is used to introduce new features into the population but tries to not add any noise or irrelevant features. Another memetic operation is super individual creation. These individuals are created from pools of features. Up to, K_f features are selected for those individuals in a random manner and create $\alpha \cdot W$ new chromosomes. In the next step, all newly created individuals are evaluated. Evaluation calculates fitness of individuals using the validation set and returns value based on a configured metric based on the confusion matrix.

The N fittest individuals are eventually selected to maintain a steady size of the population (line 15). In the end, K_f is adaptively grown based on current fitness improvement. When fitness growth stops and the average number of features selected in chromosome is close to K_f, it is increased. The mode of parent selection is also adjusted between local and global. The algorithm starts in global mode. The mode is switched to local after the growth of K_f. If no improvements to fitness are made in the three last generations it is switched back to global. The evolution process continues until there is no improvement in the average fitness of the population greater than the $\eta_{\min}$.

3.3 Alternating algorithm for optimization of SVM hyperparameters and training set selection

Tuning the SVM model is tricky when the training set is to be reduced, because of the mutual dependence—the SVM hyperparameters are usually required to select $\boldsymbol{T'}$, while depending on the subset used for training, a different SVM model (set of hyperparameters) might be optimal. Previously the main problem of either GASVM or MASVM was the need to setup the hyperparameters of the SVM model. In previous works, the hyperparameters were determined through a GS, which is a computationally intensive process. Consequently, the practicality of the training set selection algorithm was limited, despite its primary objective of enhancing SVM's usability on large datasets. The algorithm presented in this section was introduced to test the hypothesis that $\mathcal{M}$ optimization can yield good results in combination with training set selection when compared to running the training set selection algorithm with parameters found by GS

The first solution to test this hypothesis was ALGA—**AL**ternating **G**enetic **A**lgorithm for selecting the SVM model and refining the training set [76] (refining means selecting $\boldsymbol{T'}$). The main concept of this algorithm is to run one optimization at a time but use the results propagated from the other optimization phase. When the current best solution could not be improved further, the algorithm changes its mode. This consists of saving the current best solution and switching to optimize either $\mathcal{M}$ or $\boldsymbol{T'}$. For the model optimization phase, GAHP described in Section 3.1 is used while training set optimization is based on GASVM algorithm [75]. Although the algorithm provides a modular approach to the problem, the training set selection process was afterward changed to improved MASVM (see Section 2.3.2), which does not need to specify the desired size of the training set beforehand. This improved algorithm is designated as ALMA from **AL**ternating **M**emetic **A**lgorithm.

The pseudocode of ALGA/ALMA is given in Algorithm 4. In the beginning (line 1) the population for $\boldsymbol{T'}$ (designated as P) is generated by randomly drawing vectors from the training set. This population is evaluated with predefined kernel hyperparameters in lines 3-5. Simultaneously, population representing SVM hyper-

Algorithm 4 Alternating genetic algorithm for optimization of SVM hyperparameters and training set selection

```
 1: Initialize population of $\boldsymbol{T'}$ ($P$) of size $N$
 2: Initialize population of $\mathcal{M}$ ($Q$) of size $M$
 3: $q_{init} \leftarrow$ GetDefault($Q$)
 4: for all $p_i \in P$ do
 5:     $\eta_i \leftarrow$ CalculateFitness($p_i$, $q_{\text{init}}$)
 6: $p_{\text{Best}} \leftarrow$ SelectBest($P$)            ▷ individual with the highest fitness
 7: repeat
 8:     repeat                                             ▷ $\mathcal{M}$ optimization phase
 9:         for all $q_i \in Q$ do
10:             $\eta_i \leftarrow$ CalculateFitness($p_{\text{Best}}$, $q_i$)
11:         $q_{\text{Best}} \leftarrow$ SelectBest($Q$)
12:         $Q \leftarrow$ IterateModelOptimization($Q$)
13:     until LocalStopCondition
14:     if $\neg$ GlobalStopCondition then
15:         repeat                                         ▷ $\boldsymbol{T'}$ optimization phase
16:             for all $p_i \in P$ do
17:                 $\eta_i \leftarrow$ CalculateFitness($p_i$, $q_{\text{Best}}$)
18:             $p_{\text{Best}} \leftarrow$ SelectBest($P$)
19:             $P \leftarrow$ IterateTrainingSetOptimization($P$)
20:         until LocalStopCondition
21: until GlobalStopCondition
22: return ($p_{\text{Best}}$, $q_{\text{Best}}$)
```

parameters (Q) is created (line 2). After fitness values are obtained for those initial hyperparameters, the best individual in P is selected (p_{Best}). Starting in line 8, the algorithm switches to model optimization phase. First of all the whole population Q is evaluated. What is important, evaluation is performed on $\boldsymbol{T'}$ (p_{Best}). This helps to save a lot of computation time. In each iteration, the best-performing hyperparameters set is saved (line 11). In line 12, all of the genetic operators are applied, a new population is created, evaluated and the selection process is performed (all of those operations are from GAHP algorithm). The model optimization phase iterates until the algorithm used for model optimization stops. In the case of GAHP, the stop condition is based on the growth of the average fitness of all individuals in the population by more than η_{min}. So local stop condition is based on the optimization method selected. It is used to decide when to switch the optimization phases.

Afterwards algorithm switches its mode to optimize $\boldsymbol{T'}$ (lines 15-20). The same logic is applied here, first, the whole population is evaluated using the best set of hyperparameters from the previous phase (q_{Best}). Currently best $\boldsymbol{T'}$ is saved in p_{Best} after which one iteration of applying genetic operators of GASVM (or MASVM in case of ALMA) is performed. The local stop condition is the same as with the $\mathcal{M}$ optimization phase (so both phases use the same local stop condition). This alternating process is repeated as long as at least one of two subsequent phases manages to improve the average fitness. This means that the algorithm needs to run at least the model optimization phase and the instance selection process (phases goes: $\mathcal{M} \rightarrow \boldsymbol{T'}$). Otherwise, if any of the phases cannot further improve the solution, the global stop condition is reached and the SVM trained with the best individuals (p_{Best}, q_{Best}) is returned.

This algorithm was first presented in [76] and was later extended to include the feature selection process. In the beginning, the feature selection was done in a preprocessing step which filtered the feature set. Afterwards, the alternating process was extended to add the third phase with the evolutionary feature selection algorithm presented in Section 3.2. This extended alternating algorithm was published in [37]. All of that development led to the creation of the simultaneous optimization algorithm (presented in the next section).

3.4 Simultaneous optimization of training and feature set and SVM hyperparameters

The problems of optimizing training data ($\boldsymbol{T'}$ and $\boldsymbol{F'}$) and $\mathcal{M}$ are mutually dependent on each other. Hence, the previously presented alternating approach might not be an optimal solution to that problem. It could be easily imagined that performing feature set optimization under suboptimal $\boldsymbol{T'}$ or $\mathcal{M}$ could lead to obtaining results that are stuck in local optima for $\boldsymbol{F'}$ (as the search space is very large) or could in extreme cases deteriorate optimization and lead to poor performance of such model (where first bad $\mathcal{M}$ is selected which impacts subsequent phases of the algorithm). Therefore, an algorithm to simultaneously optimize all three components (the $\boldsymbol{T'}$ and $\boldsymbol{F'}$ and $\mathcal{M}$) is proposed and denoted as SE-SVM—**S**imultaneous **E**volutionary optimization of **SVM**. This algorithm allows for verifying the first research hypothesis that such simultaneous optimization could provide improved training and classification time without affecting classification quality. This means that during the tests this algorithm should provide a Pareto-optimal solution regarding the training and classification times coupled with proper classification metrics. By improving upon the previously shown alternating scheme it is also hypothesized that it should improve classification quality in relation to ALMA method. One of the advantages of this algorithm is starting with relatively small training sets (thanks to training set selection) and coupling them with hyperparameters. This should help to quickly evaluate initial solutions and explore more prominent regions of the search space. Moreover, the ability to adapt (grow) the size of selected $\boldsymbol{T'}$ and $\boldsymbol{F'}$ should keep the good quality of classification. However, adding feature selection could increase the computation time compared to ALMA or MASVM methods, so it is not expected to outperform these methods in absolute computation time but rather provide a Pareto-optimal solution. This algorithm will also be studied on 2D dataset where feature selection is turned-off (as it does not make sense to test it there).

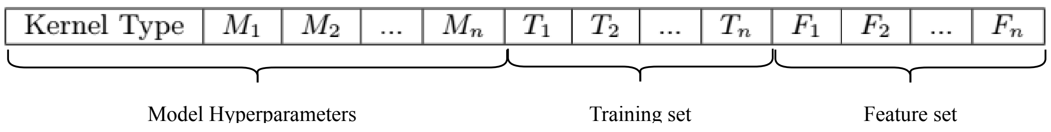


Figure 3.2: The design of chromosome for SE-SVM method. Source: [38].

The design of the chromosome for SE-SVM is shown in Figure 3.2. There are three distinct parts. The hyperparameters and kernel type make up the first part, which has a constant size and can hold any number of arbitrary selections for the kernel type. The latter two parts consist of sets for training vectors and features. The IDs are representing numbers of selected rows (samples) and columns (features), and are maintained unique throughout the evolution process. IDs are being used instead of binary encoding due to the impracticality for large datasets. Furthermore, those sets are expected to grow during the evolution process starting from some small values of K_f for $\boldsymbol{F}$ and K_t for $\boldsymbol{T}$. Hence the size of each chromosome is dynamic and can differ as K_f and K_t can grow independently. Note that not even all parts of $\boldsymbol{T'}$ or $\boldsymbol{F'}$ in a given population need to have the same size.

The pseudocode of this algorithm is presented in Algorithm 5. Although there is only a single population, the parts of chromosomes population regarding $\mathcal{M}$ will be referred to with P^m, for $\boldsymbol{T'}$ as P^t and for $\boldsymbol{F'}$ as P^f.

Algorithm 5 Pseudocode for simultaneous evolutionary optimization of SVM algorithm

```
1:  $P_{\text{best}} \leftarrow \emptyset$, $\mathcal{S}_{\text{pool}} \leftarrow \emptyset$, $\mathcal{F}_{\text{pool}} \leftarrow \emptyset$
2:  Initialize population $P$ of size $N$
3:  for i = 0 to 5 do
4:      for all $p_i \in P$ do
5:          $\eta_i \leftarrow$ CalculateFitness($p_i$)
6:      repeat
7:          $\mathcal{S}_{\text{pool}}, \mathcal{F}_{\text{pool}} \leftarrow$ UpdatePools($P$)
8:          Parents $\leftarrow$ SelectParents($P$)
9:          $P' \leftarrow$ Crossover(Parents)
10:         Education($P'^t$, $P'^f$, $\mathcal{S}_{\text{pool}}$, $\mathcal{F}_{\text{pool}}$)
11:         Mutation($P'$)
12:         $P'_{SI} \leftarrow$ SuperIndividualsCreation($\mathcal{S}_{\text{pool}}, \mathcal{F}_{\text{pool}}$)
13:         for all $p_i \in (P \cup P'_{SI})$ do
14:             $\eta_i \leftarrow$ CalculateFitness($p_i$)
15:         $P \leftarrow$ Selection($P \cup P'_{SI} \cup P'$)
16:         Adaptation($P$)
17:     until StopCondition
18:     $P_{\text{best}} \leftarrow P_{\text{best}} \cup$ GetBest($P$)
19:     $P \leftarrow$ RegeneratePopulation
20: return Best($P_{\text{best}}$)
```

The algorithm starts by creating a new population (line 2) using different initialization methods for each part of the chromosome. As this algorithm is built over previous solutions the details of each operator could be found in proper sections: for kernel evolution, GAHP is utilized (see Section 3.1), for training set selection the MASVM is used (see Section 2.3.2) and for feature selection the EFS method is used (see Section 2.3.3). Afterwards, the population is evaluated and each individual has their fitness value calculated. In line 7, the pools of features and SVs are being updated based on the current population. These pools hold information about the features ($\mathcal{F}_{\text{pool}}$) and vectors ($\mathcal{S}_{\text{pool}}$) that are discovered to be valuable during evolutionary process, and later, help to exploit knowledge learned while running computations. During the first iteration, this is done based on the fitness and SVs of the initial population. In the next step, parents are selected using a local-global adaptation scheme (Section 2.3.2). This works similarly to EFS and MASVM, the algorithm starts in global mode. The population is divided into two parts, by sorting individuals based on their fitness and ϵ threshold. In global mode, parents are selected from different parts of the population while in the local mode, both are selected within the same group. The crossover operator (line 9), creates N new individuals which are joined into a new population (P') of the same size as the original one. Here proper operators are applied over each part of the chromosome. The kernel type is kept constant during the evolution process where the hyperparameters are crossed over using Equation 3.1 and sets of $\boldsymbol{T'}$ and $\boldsymbol{F'}$ from two individuals are summed separately. The size of these sets is limited by the current values of K_t for $\boldsymbol{T'}$ and K_f for $\boldsymbol{F'}$. If either of those sets grows bigger, then a random selection process is employed to maintain proper size. In the next line, the parts of the population representing $\boldsymbol{T'}$ and $\boldsymbol{F'}$ (P^t and P^f respectively) are going through the operation of education. This operation involves using previous information learned during the evolution to improve the solution. For each type of set a pool of promising vectors (built-up from SVs—$\mathcal{S}_{\text{pool}}$) and features (built by creating a histogram of features used in best-fitted individuals—$\mathcal{F}_{\text{pool}}$) is used to replace vectors and features present in the chromosome. Afterwards, in line 11 whole new population goes through the process of mutation. When a chromosome is chosen for mutation, all three parts undergo this process, and each segment (part of a chromosome) employs a uniquely designed operator for $\mathcal{M}$ (GAHP), $\boldsymbol{T'}$ (MASVM), and $\boldsymbol{F'}$ (EFS). Before the evaluation and selection process, super individuals are created. This process selects up to K_f and K_t from the $\mathcal{F}_{\text{pool}}$ and $\mathcal{S}_{\text{pool}}$ for $\boldsymbol{F'}$ and $\boldsymbol{T'}$ respectively. For the model hyperparameters, the current best solution is selected and added to all new super individuals, so they share common $\mathcal{M}$. In the next step, all of the newly created individuals are trained and their fitness is calculated (based on $\boldsymbol{V}$). Training of SVM models involves selecting proper training vectors based on $\boldsymbol{T'}$ of the chromosome, as well as filtering the set of features based on $\boldsymbol{F'}$. Each SVM should have a unique training set. The selection process keeps the constant size of the population by selecting N fittest individuals. At the end of each loop, the K_f and K_t can be grown adaptively and the mode for parent selection could be adjusted. The process of evolution lasts till there is no improvement in the average fitness of the population. Here the process of regeneration is introduced once again and always R regeneration are performed. Before each regeneration, the current best solution is saved and stored in P_{best}. The result of the algorithm is the best model from the entire evolutionary process selected from P_{best}.

3.5 Adaptive RBF kernel

Although the previous solutions already optimized training data and SVM hyperparameters, it may be hypothesized that a custom kernel function can further improve the SVM capabilities—this observation is built upon visualization of multiple SVM models using 2D datasets, where different values of γ (used in RBF kernel) provided insights into how far a given SV is affecting the decision hyperplane. Especially at the start of EAs, like ALMA or SE-SVM, where there were only K_t vectors selected most of those SVs provided classification information only in their neighborhood. Thus, this observation led to the hypothesis that selecting different kernel hyperparameters for different training vectors may help better reflect the subtle characteristics of the space while determining the hyperplane. This of course brings quite a big optimization problem where the number of hyperparameters is equal to $|\boldsymbol{T}|$. In this approach, the $\boldsymbol{T'}$ optimization is joined with $\mathcal{M}$ in a form of a single algorithm, as each training vector is coupled with its own γ value for the RBF kernel. This algorithm focuses on RBF kernels with a single hyperparameter γ (see Table 2.1), because of their satisfactory performance demonstrated in many cases [88]. This algorithm partly relates to the first hypothesis that simultaneous optimization should improve classification and training time without affecting the classification quality. However, compared to SE-SVM it is expected to improve classification performance while training time is expected to grow (as the optimization problem of selecting $\boldsymbol{T'}$ is bigger). So it is expected to provide a Pareto-optimal solution with the shift toward classification quality.

It can be observed that the value of the RBF function is dependent on the

Euclidean distance between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, where the closer the vectors are in the input space, the larger the value of the kernel becomes. The hyperparameter γ is used to control the "range" of influence of SVs. By attaching γ to training vectors, a much more detailed decision boundary could be produced. This situation is presented in Figure 3.3 where an example of using different γ values coupled to the $\boldsymbol{T}$ vectors that are selected as SVs is rendered. As presented with the orange SVs in Figure 3.3b (those vectors have $\gamma = 10^5$) the larger value of this hyperparmeter results in a small radius of influence on decision hyperplane. What is more, those SVs are located near the decision boundary in regions that could be considered "difficult" to classify. These regions could also be characterized as ones where vectors from $\boldsymbol{T}$ lay close to one another. The SVs with relatively large γ values provide fine details about this "local" part of the dataset. Contrary to them SVs with small γ's (blue and red ones) tend to lay further from the decision boundary and appear in more homogeneous regions of the dataset which can be considered as "easier" for correct classification.

Please note that the number of SVs is just a small subset of the training data (in Figure 3.3 each pixel is a vector in the dataset, so presented white and black dots actually includes tens of unique data vectors while SVs are just single pixels). This sample proved that SVMs could be further improved by not only evolving $\boldsymbol{T'}$ and its $\mathcal{M}$ but also joining it with an adaptive kernel. This new method is called **SVMs** with **A**daptive **RBF** kernels—ARBF-SVM.

As discussed in Section 2.1.2 the kernel function needs to fulfill Mercer's conditions [62]. This could also be stated that the kernel matrix (or Gram matrix) must always be positive semi-definite (also the kernel function needs to be symmetric). The work presented here provides an experimental approach, where those requirements might not always be satisfied. However, it does not mean that such SVM model would not work or SVM training could not be done. By allowing relaxation of the conditions we lose the guarantee that provided decision boundary will have the maximal possible margin. This is due to the fact that the optimization problem might no longer be a convex problem, hence local minimum could be different from the global minimum. The fulfilment of those conditions was tested by checking during the evaluation process in the mentioned algorithm if all of the kernel matrices were PSD. Although it was not true in all of the cases, the Gram

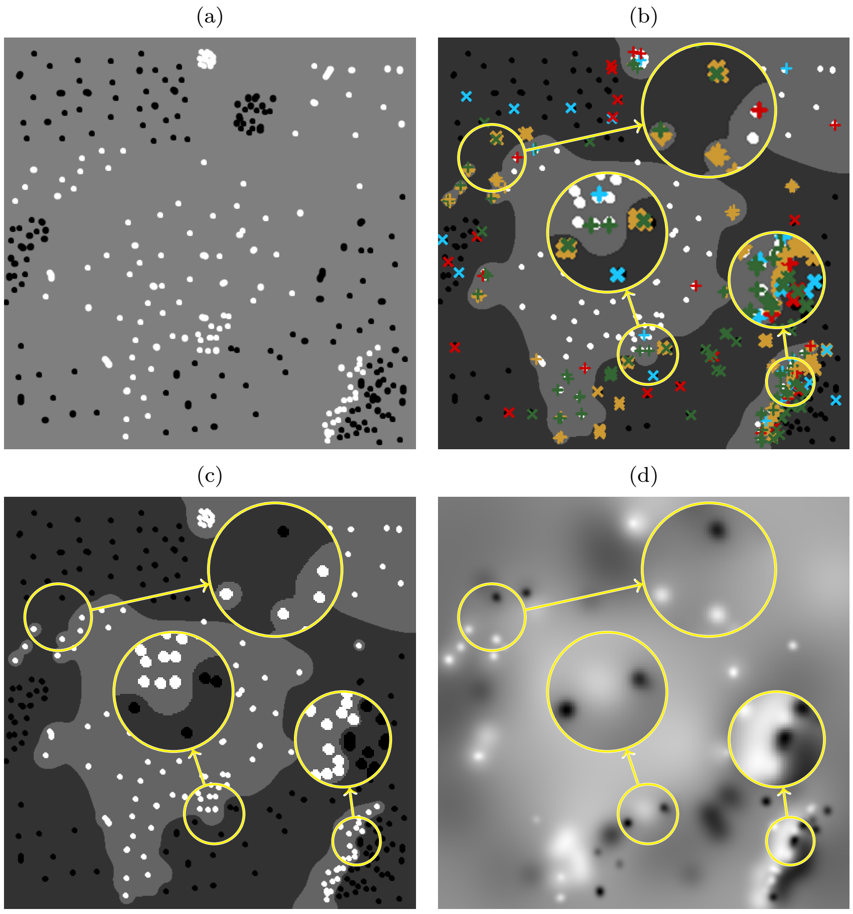


Figure 3.3: Assigning different γ's in the RBF kernel to different $\boldsymbol{T}$ vectors can help better "model" the peculiarities of the SVM hyperplane: (a) our example set (we consider binary classification, white and black pixels present examples from two different classes), (b) the crosses in different colours render SVs with different γ's: $\gamma = 10^2$ (blue), $\gamma = 10^3$ (red), $\gamma = 10^4$ (green), and $\gamma = 10^5$ (orange), together with (c) the different shades of gray show the decision boundary, (d) the "range of influence" of all SVs (the brighter/darker the pixel is, the distance to the hyperplane is larger). The SVM regularization parameter was grid-searched and was $C = 10$. This figure comes from our paper [94].

matrix was positive semi-definite in the majority of cases. As shown in practice (see Section 4.2.2) this relaxation of conditions for kernel function does not deteriorate the performance of the final classifier.

One more important step in the problem of capturing fine-grained characteristics of the training set using different γ values assigned to the $\boldsymbol{T}$ vectors is how to aggregate those γ's during the calculation of the kernel matrix. Therefore, it is important to determine a suitable aggregation function to calculate the kernel value between $\boldsymbol{x_i}$ and $\boldsymbol{x_j}$, since they may have different values of γ_i and γ_j. The following aggregation functions are examined, resulting in various versions of the evolutionary algorithm being proposed (they will be discussed later in this section):

$$\mathcal{K}(\boldsymbol{x_i}, \boldsymbol{x_j}) = e^{-\gamma_i ||\boldsymbol{x_i}-\boldsymbol{x_j}||^2} \qquad \gamma\text{-One} \qquad (3.3)$$
$$\mathcal{K}(\boldsymbol{x_i}, \boldsymbol{x_j}) = e^{-[(\gamma_i+\gamma_j)/2] ||\boldsymbol{x_i}-\boldsymbol{x_j}||^2} \qquad \gamma\text{-Avg} \qquad (3.4)$$
$$\mathcal{K}(\boldsymbol{x_i}, \boldsymbol{x_j}) = e^{-max(\gamma_i,\gamma_j) ||\boldsymbol{x_i}-\boldsymbol{x_j}||^2} \qquad \gamma\text{-Max} \qquad (3.5)$$
$$\mathcal{K}(\boldsymbol{x_i}, \boldsymbol{x_j}) = e^{-min(\gamma_i,\gamma_j) ||\boldsymbol{x_i}-\boldsymbol{x_j}||^2} \qquad \gamma\text{-Min} \qquad (3.6)$$
$$\mathcal{K}(\boldsymbol{x_i}, \boldsymbol{x_j}) = e^{-\gamma_i ||\boldsymbol{x_i}-\boldsymbol{x_j}||^2} + e^{-\gamma_j ||\boldsymbol{x_i}-\boldsymbol{x_j}||^2} \qquad \gamma\text{-Sum} \qquad (3.7)$$

It should be noted that in the γ-One variant, a single γ is selected and therefore, aggregation of these hyperparameters is not carried out, as demonstrated in our previous work regarding adaptive kernels [94]. It is worth mentioning that γ could be interpreted as the range of influence of a given SV. Finally, once the training of an SVM is completed, the calculation of the kernel for the new test sample ($\boldsymbol{x_{\mathrm{test}}}$) is performed in the following manner:

$$\mathcal{K}(\boldsymbol{x_{\mathrm{SV}}}, \boldsymbol{x_{\mathrm{test}}}) = e^{-\gamma_{\mathrm{SV}} ||\boldsymbol{x_{\mathrm{SV}}}-\boldsymbol{x_{\mathrm{test}}}||^2}, \qquad (3.8)$$

where $\boldsymbol{x_{\mathrm{SV}}}$ is an SV and γ_{SV} is γ value associated with this SV.

The algorithm starts with an initial preprocessing step performed over a random balanced subset (of size $c \cdot K_t$) of the entire training set $\boldsymbol{T}$, where c represents the number of classes in the input dataset, K_t is a hyperparameter of the method ($K_t \leq |\boldsymbol{T}_{\min}|$), and $|\boldsymbol{T}_{\min}|$ is the number of training vectors in the least numerous class. The goal is to obtain a single C value (that will remain unchanged throughout the optimization process), together with γ that will be used to generate a set of γ's which are used in the evolutionary process (Algorithm 6 line 4).

For each $\gamma_i \in \vec{\gamma}$, an initial population P of N individuals is generated (p_j, $j = 1, 2, \ldots, N$) which is evolved (line 20). Each chromosome p_j encompasses a

Algorithm 6 Evolving reduced training sets with adaptive γ values using DA-SVM.

```
 1: function FindGammaAndC
 2:     $(C, \gamma) \leftarrow$ Grid search over a random $\boldsymbol{T'}$ of size $c \cdot K$
 3:     $\vec{\gamma} \leftarrow \{\gamma/10, \gamma, 10 \cdot \gamma, 100 \cdot \gamma, 1000 \cdot \gamma\}$
 4:     return $\vec{\gamma}, C$

 5: function EvolveT($P$, $\mathcal{S}_{\text{best}}$)
 6:     $\mathcal{S}_{\text{pool}} \leftarrow \emptyset$
 7:     while termination condition not met do
 8:         $P' \leftarrow$ Crossover($P$)
 9:         $P' \leftarrow$ Educate($P'$)
10:         $P' \leftarrow$ Mutate($P'$)
11:         $P' \leftarrow$ Calculate fitness($P'$, $\mathcal{S}_{\text{best}}$)
12:         $\mathcal{S}_{\text{pool}} \leftarrow$ Update SV pool($P'$)
13:         $P^{\text{SI}} \leftarrow$ Create super individuals($\mathcal{S}_{\text{pool}}$)
14:         $P \leftarrow$ Post select($P$, $P'$, $P^{\text{SI}}$)
15:         $P'_{\text{best}} \leftarrow$ Find best individual($P$)
16:         Adapt($P$)
17:     return $P$, $P'_{\text{best}}$
18: function Optimize($\vec{\gamma}, C$, $P'_{\text{best}}, P_{\text{best}}, \mathcal{S}_{\text{best}}$)
19:     for all $\gamma_i$ in $\vec{\gamma}$ do ▶ $\gamma$'s sorted ascendingly
20:         $P \leftarrow$ Generate($N$, $C$, $\gamma_i$, $\boldsymbol{T}$)
21:         $P'_{\text{best}} \leftarrow$ Find best individual($P$)
22:         if $\eta(P'_{\text{best}}) > \eta(P_{\text{best}})$ then
23:             $P$, $P'_{\text{best}} \leftarrow$ EvolveT($P$, $\mathcal{S}_{\text{best}}$)
24:             if $\eta(P'_{\text{best}}) > \eta(P_{\text{best}})$ then
25:                 Add SV($P'_{\text{best}}$) to $\mathcal{S}_{\text{best}}$
26:                 $\boldsymbol{T} \leftarrow$ Shrink($\boldsymbol{T}$, $P$)
27:                 $P_{\text{best}} \leftarrow P'_{\text{best}}$
28:         Reset LGA, $\mathcal{S}_{\text{pool}} \leftarrow \emptyset$
29:     return $P_{\text{best}}$

30: function main
31:     $P'_{\text{best}} \leftarrow \emptyset$, $P_{\text{best}} \leftarrow \emptyset$, $\mathcal{S}_{\text{best}} \leftarrow \emptyset$
32:     $\vec{\gamma}, C \leftarrow$ FindGammaAndC
33:     Optimize($\vec{\gamma}$, $C$, $P'_{\text{best}}$, $P_{\text{best}}$, $\mathcal{S}_{\text{best}}$)
```

(potentially imbalanced) subset of $\min\{K, |\boldsymbol{T}_c|\}$ training vectors, alongside their gamma values set to γ_i, randomly sampled from $\boldsymbol{T}$ for each class c. Once the population is created, the *fitness* of all individuals is calculated. For each p_j, an SVM is trained using a combined set of $\boldsymbol{T}'_j$ and $\mathcal{S}_{\text{best}}$ vectors, where $\boldsymbol{T}'_j$ is the encoded reduced set of p_j, and $\mathcal{S}_{\text{best}}$ is a set of all vectors that have been selected as SVs for the *best* individuals in the previous generations (for the first γ_i, $\mathcal{S}_{\text{best}} = \emptyset$).

The fitness η_j of the *j-th* chromosome is calculated as AUC obtained using the corresponding SVM over $\boldsymbol{V}$. If η of the best individual P'_{best} (line 21) is greater than η of the best individual P_{best} evolved so far (line 22), appending more $\boldsymbol{T}$ vectors with larger γ's will help enhance the abilities of the SVM through more fine-grained modeling of the hyperplane. In that case, the refined sets are optimized using a memetic algorithm [96] (line 23). Otherwise, γ_i is skipped, and the algorithm jumps to the next γ.

Note that calculating the fitness value requires training an SVM using the corresponding $\boldsymbol{T}'$, therefore this algorithm is a *wrapper approach*. Since the training process is the most time-consuming part of the entire algorithm, it may be considered as an important drawback of the technique (although the reduced training sets are kept small). To further tackle this issue, the No-T variant is proposed in which training of an SVM is *not* performed for calculating the fitness of an individual. Instead, *all training vectors are selected as SVs*, hence build upon observation that the evolved training sets should encompass only important training vectors that are likely to be SVs. This approach may, however, slow down the classification proccess of an underlying SVM which is in a linear relation with the number of SVs.

Once the evolution over the *i-th* gamma is finished, it is verified if the fitness of the best individual from the final population is greater than the fitness of the best solution already found (line 22). If so, the SVs (together with the corresponding γ's) of the fittest individual are added to $\mathcal{S}_{\text{best}}$, being the pool of SVs of the best individuals (line 25). These vectors are removed from $\boldsymbol{T}$ which becomes $\boldsymbol{T} \leftarrow \boldsymbol{T} - \text{SV}(P'_{\text{best}})$, and—additionally—the training set is shrunk and the $\boldsymbol{T}$ vectors that have been correctly classified by all individuals in the population are removed (line 26), as they are often positioned far from the decision hyperplane. This reduction process can considerably decrease the size of $\boldsymbol{T}$, hence enhance the convergence abilities of the evolutionary optimization, because discarded examples are unlikely to be picked as SVs. It, however, may lead to extremely imbalanced $\boldsymbol{T}$'s (even containing single-class vectors). To tackle this issue, the $\mathcal{S}_{\text{best}}$ vectors are incorporated in the final reduced training sets. Finally, the best individual P_{best} is returned (line 29).

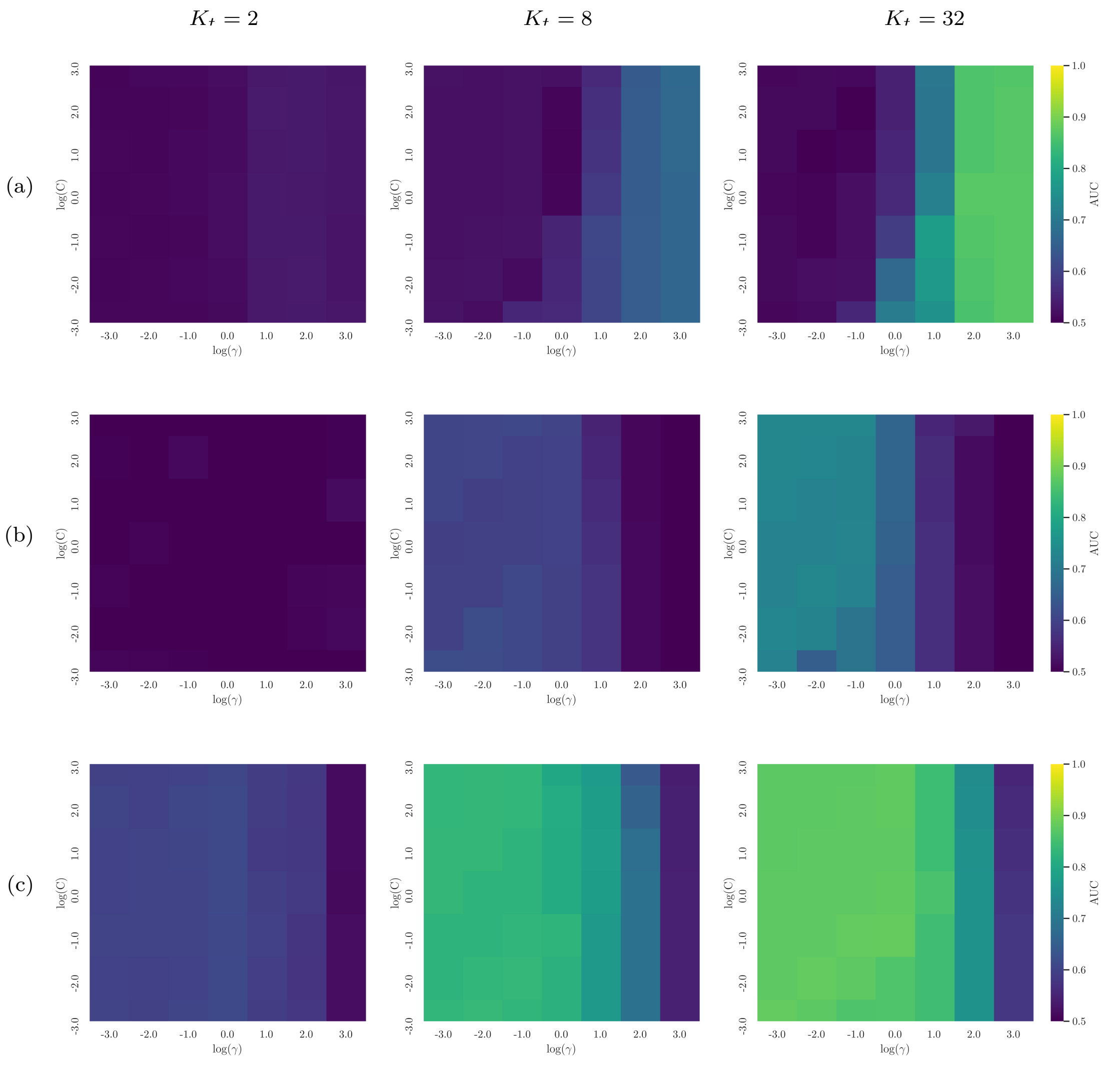


Figure 3.4: The impact of the K_t value on the quality of hyperparameter configurations extracted for example datasets: (a) 2D-Blobs, (b) german, (c) spambase. For the details of these sets (their sizes and characteristics), see Section 4.1. This figure comes from our paper [40]

This preliminary grid search (with a logarithmic step)[1] is based upon the

[1]Note that the grid search may be easily replaced with other algorithms, e.g., a selected

observation that the small subsets of the entire training set are often representative enough to locate high-quality hyperparameter configurations [77]. In Figure 3.4, the impact of the K_t values is visualized on the estimated quality of the hyperparameter configuration. For the performance measure AUC is calculated for the *validation* set ($\boldsymbol{V}$) using an SVM trained with a selected refined set. The three example datasets are selected for this investigation (the details of these sets are reported in Section 4.1). What can be appreciated is that $K_t = 8$, being the value of K_t used in as default value in the algorithm, is enough to observe the dependency between the (γ, C) pairs and the corresponding AUC obtained using an SVM trained over such reduced sets (note that $K_t = 2$ may be too small for some datasets, e.g., 2D-Blobs in this example). On the other hand, larger K_t's, here $K_t = 32$, do not bring significant improvements in the estimation quality of the underlying hyperparameter pairs (the shape of the solution space in Figure 3.4 remains similar to $K_t = 8$), but adversely affect the SVM training time.

This initial method was refined and improved by DA-SVM method, where different kernel functions are combined in a single SVM. Figure 3.5 renders an example dataset (2D-Linear pool) which could benefit from exploiting a linear kernel in combination with the RBF one. Although an SVM with an adaptive RBF kernel (our ARBF-SVM method [94]) is able to effectively elaborate a high-quality decision hyperplane, the number of SVs is significantly larger than for a classifier which initially determines a linear hyperplane, and then fine-tunes it with RBF kernels. Also, since the hyperplane seems less "overfitted" to the training set, it could potentially deliver better generalization.

Joining of the kernels is based on the observation that the kernels can be divided into the *global* and *local* ones [126]. In the former case, data points that are far away from the test point may have a significant effect on the kernel value (e.g., as in the linear kernel), whereas in the latter kernels, only those vectors which lay close to the test example impact the kernel value (e.g., as in the RBF kernel). Hence, the best way to proceed is to elaborate an "approximate" decision hyperplane using the linear kernel, and locally improve its shape with the use of adaptive RBF.

The extension of the previous approach to incorporate this design of joining

metaheuristics. It was investigated in our publication [40] but replacing this algorithm did not change the performance of the algorithm.

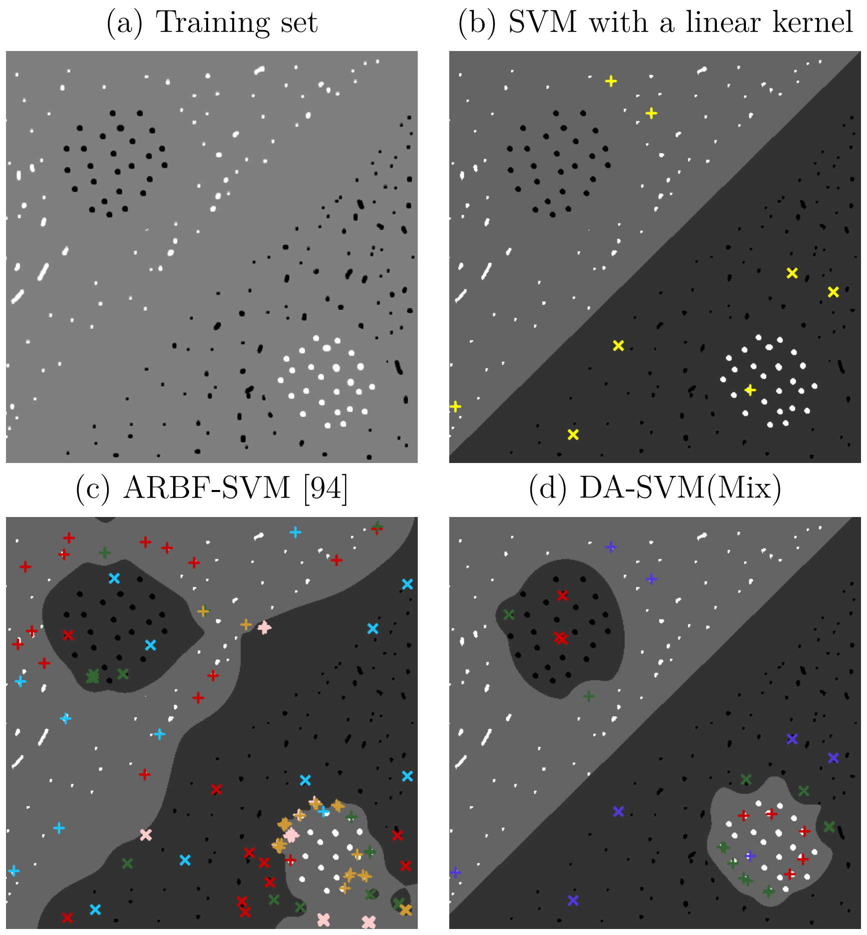


Figure 3.5: Example hyperplanes extracted using various methods show that mixing the linear and RBF kernels may help obtain "less complicated" hyperplanes (hence possibly not overfitted to $\boldsymbol{T}$). We visualize (a) an example training set (with white and black dots presenting two-class vectors), (b) a linear-kernel SVM (with yellow crosses showing the SVs), (c) a hyperplane obtained by ARBF-SVM [94] and (d) DA-SVM(Mix). In (d), the purple crosses render the SVs obtained for the linear kernel while the others visualize the SVs for various γ's for the RBF kernel. This figure comes from our paper [40]

kernels is presented in Algorithm 7. For the linear kernel, the fitness of the individuals is quantified by using the balanced accuracy (instead of AUC), in which both sensitivity and specificity equally contributes to η, and can be safely used for imbalanced classification [49] (line 13).

The initial pre-processing is performed using grid search to optimize C, and

Algorithm 7 Combining the linear and RBF kernels in a single SVM using DA-SVM(Mix).

```
1: function FindGamma($\mathcal{S}_{\text{best}}$, $C$)
2:     $\vec{\gamma} \leftarrow \{0.001, 0.01, 0.1, 1, 10, 100, 1000\}$
3:     $\vec{\gamma}_\eta \leftarrow \emptyset$
4:     for all $\gamma_i$ in $\vec{\gamma}$ do
5:         $P \leftarrow$ Generate ($N$, $C$, $\gamma_i$, $\boldsymbol{T}$, $\mathcal{S}_{\text{best}}$)
6:         $P'_{\text{best}} \leftarrow$ Find best individual($P$)
7:         $\vec{\gamma}_\eta \leftarrow \vec{\gamma}_\eta \cup \eta(P'_{\text{best}})$
8:     $\gamma \leftarrow \vec{\gamma}[\text{index}_{\max}(\vec{\gamma}_\eta)]$
9:     $\vec{\gamma} \leftarrow \{\gamma/5, \gamma, 10 \cdot \gamma, 50 \cdot \gamma\}$
10:    return $\vec{\gamma}$

11: function Main
12:    $P'_{\text{best}} \leftarrow \emptyset$, $P_{\text{best}} \leftarrow \emptyset$, $\mathcal{S}_{\text{pool}} \leftarrow \emptyset$, $\mathcal{S}_{\text{best}} \leftarrow \emptyset$
13:    Metric = Balanced Acc
14:    $C \leftarrow$ Grid search over a random $\boldsymbol{T'}$ of size $c \cdot K$
15:    $P \leftarrow$ Generate($N$, $C$, $\boldsymbol{T}$)
16:    $P, P'_{\text{best}} \leftarrow$ EvolveT($P$, $\mathcal{S}_{\text{best}}$) ▶ as in Algorithm 6
17:    Add SV($P'_{\text{best}}$) to $\mathcal{S}_{\text{best}}$
18:    $\boldsymbol{T} \leftarrow$ Shrink($\boldsymbol{T}$, $P'_{\text{best}}$)
19:    $P_{\text{best}} \leftarrow P'_{\text{best}}$
20:    Metric = AUC, $K_t = 2 \cdot K_t$
21:    $\vec{\gamma} \leftarrow$ FindGamma($\mathcal{S}_{\text{best}}$, $C$)
22:    return Optimize($\vec{\gamma}, C$, $P'_{\text{best}}, P_{\text{best}}, \mathcal{S}_{\text{best}}$) ▶ as in Algorithm 6
```

evolve a population of refined training sets with linear kernels *only*. When the evolution ends, all SVs with the linear kernel is stored in $\mathcal{S}_{\text{best}}$ and utilized later in the optimization process (line 17). Afterwards, the shrink process is applied, but with a modification that all the training vectors correctly classified by the best individual are removed. This approach may be considered as "more aggressive", as the size of $\boldsymbol{T}$ will decrease much faster in such a setting. Once the process of optimizing the linear SVM has been finalized, the fitness is switched back to AUC (line 20), and K_t is doubled to make capturing the more "difficult" parts of the space using an RBF kernel easier. Then a process of finding $\vec{\gamma}$ for adaptive RBF kernel is performed. The random populations for each γ value (line 2) are created and evaluated. The best-fitted individual (line 6) is extracted and its fitness is stored in $\vec{\gamma}_\eta$ (line 7). At this point, SVMs may already contain vectors with both linear and RBF kernels as $\mathcal{S}_{\text{best}}$ is utilized. Afterwards, the γ value with the largest corresponding fitness is found (line 8) and $\vec{\gamma}$ is build upon it (line 9). Finally, the $\boldsymbol{T'}$ sets are evolved in the same way as presented in Algorithm 6.

The one disadvantage of such an approach is that it is hard to know if a given dataset could be "coarsely" modeled, by linear SVM, and forcing this kind of solution may be harmful to the end result. It was proven in initial experimentation that this is true for some of the benchmark datasets where this algorithm worsens the performance of the model. That is why a co-evolution scheme is proposed in Algorithm 8 to combine these two approaches and benefit from the concurrent optimization of different kernel combinations. In order to tackle the computational cost a competitive-like approach [118, 121] is used in which the better (sub-)populations trigger the process of pruning the worse populations that are "outperformed".

First, a separate linear-kernel population P^i of size N is generated for each $C_i \in \vec{C}_{\text{L}}$ (lines 1–4). Each population P^i undergoes the evolution presented in Algorithm 6—each P^i has its own SV pool $\mathcal{S}^i_{\text{best}}$, and for each population a separate $\vec{\gamma}_i$ is determined (see Algorithm 7, line 1 for more details). Note the γ values may be dependent on the selected C, hence they are likely to be different across the populations. For each P^i, $\boldsymbol{T}$ is shrunk (Algorithm 8, line 6) using the procedure introduced in Algorithm 7 (based on the best individual performance). Afterwards, all populations are added to the pool G (Algorithm 8, line 7).

At this point, all of the linear-kernel SVMs are evolved, together with the corresponding $\vec{\gamma}_i$, $\mathcal{S}^i_{\text{best}}$, and $\boldsymbol{T}^i_P$ for each of $P^i \in G$. In the next step a separate population P^γ is generated (lines 8–9) that will optimize the RBF kernels, as in Algorithm 6. For P^γ, its training set becomes $\boldsymbol{T}$. Additionally, all of the current best individuals $\tilde{P^i}_{\text{best}}$ for each population are stored (line 11)—it will allow checking if those should be optimized at a later stage of the algorithm.

In the co-evolutionary part of the algorithm, G populations are evolved which exploit their unique training sets $\boldsymbol{T}^i_P$. To prune the entire set of all populations, the most promising ones are selected for evolution. To equally capture the quality of the entire population, alongside the quality of the best individual, the populations are sorted according to the sum of their average $\eta(P^i)$ and best $\eta(P^i_{\text{best}})$ fitness scores (lines 16–18).

Algorithm 8 Combining Algorithms 6 and 7 into a competitive co-evolutionary scheme.

```
1: $\vec{C}_{\mathrm{L}} = \{0.01, 0.1, 1, 10, 100\}$, $G \leftarrow \emptyset$
2: for all $C_i$ in $\vec{C}_{\mathrm{L}}$ do
3:     $P^i \leftarrow$ Generate($N$, $C_i$, Linear Kernel, $\boldsymbol{T}$)
4:     $P^i \leftarrow$ EvolveT($P$, $\mathcal{S}^i_{\text{best}}$)
5:     $\vec{\gamma} \leftarrow$ FindGamma($\mathcal{S}^i_{\text{best}}$, $C_i$)
6:     $\boldsymbol{T}^i_P \leftarrow$ Shrink($\boldsymbol{T}$, $P^i_{best}$)
7:     Add $P^i$ to $G$
8: $P^\gamma \leftarrow$ Generate($N$, $\boldsymbol{T}$, FindGammaAndC)
9: Add $P^\gamma$ to $G$
10: for all $P^i$ in $G$ do
11:     $P^i_{\text{best}} \leftarrow \tilde{P^i}_{\text{best}}$
12: while all $P^i$ in $G$ not finished do
13:     for all $P^i$ in $G$ do
14:         $P^i \leftarrow$ Generate ($N$, $\boldsymbol{T}^i_P$, $P^i$)
15:         $P^i \leftarrow$ Calculate fitness($P^i$, $\mathcal{S}^i_{\text{best}}$)
16:         if $|G| > 3$ then
17:             Sort $G$ by $\eta(P^i) + \eta(P^i_{\text{best}})$
18:             $G \leftarrow$ Select three best $P^i$ in $G$
19:         if $\eta(\tilde{P^i}_{\text{best}}) > \eta(P^i_{\text{best}})$ then
20:             EvolveT($P^i$, $\mathcal{S}^i_{\text{best}}$)
21:         Sort $G$ by $\eta(P^i)$ ▶ descendingly
22:         for all i in range(1, $|G|$) do
23:             $|P^i| \leftarrow |P^i| - 2$
24:             if $P^i$ size $\leq 2$ then
25:                 $G \leftarrow G - P^i$
26:         if $\eta(\tilde{P^i}_{\text{best}}) \geq \eta(P^i_{\text{best}})$ then
27:             Add SV($\tilde{P^i}_{\text{best}}$) to $\mathcal{S}^i_{\text{best}}$
28:             $\boldsymbol{T}^i_P \leftarrow$ Shrink($\boldsymbol{T}$, $P^i$)
29:             $P^i_{\text{best}} \leftarrow \tilde{P^i}_{\text{best}}$
30: return Best individual from all $P^i \in G$
```

Afterwards, the training sets for each population are optimized using the memetic algorithm (line 20). Once the evolutions are finished, the populations are sorted descendingly (line 21) and gradually decrease the size $|P^i|$ of the "weaker" populations through removing random individuals, excluding the best-fitted one (line 23). The pruning of the populations that are less promising helps to reduce the computational cost of the optimization. If the population is smaller than two chromosomes, then the population is excluded from further processing (line 25). For each population, it is verified if the best fitness P^i_{best} has improved—in this case, $\mathcal{S}^i_{\mathrm{best}}$ is updated with its SVs, and $\boldsymbol{T}$ shrunk for this population (line 28). Otherwise, if there is no fitness improvement, those vectors selected for γ_i are discarded and progress to the next γ. Ultimately, the best individual among all populations is returned (line 30). Only alive populations are considered.

In [40], many different variants of the algorithm were proposed and tested. Using this knowledge in this work the most successful ones are presented later in experimental validation. Those variants will be denoted using following naming convention **DA-SVM ([FS], [CE], Kernel aggregation, [No-T])**, where $[\cdot]$ is an optional parameter. These possible variants are:

- **FS**—if this optional parameter is present, then the feature selection (i.e., the recursive feature elimination [39]) is switched on, and it is performed before the evolutionary optimization (as a pre-processing step).
- **CE**—if this optional parameter is present, then the co-evolutionary optimization is performed.
- **Kernel aggregation**—this mandatory parameter indicates which aggregation function was used for handling multiple γ values in the RBF kernel.
- **No-T**—if this optional parameter is present, then all training vectors are selected as SVs, hence no SVM training is performed while calculating the fitness of individuals in a population.

The most successful variants included ones that use co-evolutionary optimization joined with feature selection (based on result in our paper [40]). The No-T variant is interesting and also was present among top configuration to consider. Because of that other variants will not be presented in experiments.

3.6 Building ensembles

Building an ensemble of multiple classifiers is often considered the last step toward a possible improvement of classifier performance. This popular paradigm is based on the assumption of leveraging the strength of individual classifiers and mitigating their weaknesses [112]. This opens an interesting direction that could help further improve the performance and help to deal with large datasets (as one can be using multiple lightweight SVM in the ensemble).

Based on the previous experience of designing SE-SVM and DA-SVM and experimenting with the 2D datasets, it was hypothesized that datasets often consist of homogeneous and heterogeneous regions. These regions provide different "difficulty" levels for classification, whereas homogeneous regions are easy and straightforward to classify correctly. Based on that observation, building ensemble should reassemble these different regions, so the SVM classifier should split the dataset accordingly. This initial idea resulted in the introduction of competence regions for each base model in the ensemble. The trained SVM classifier is used to split the input space into *certain* and *uncertain* parts, where certain regions are expected to provide better classification quality. A certain region is understood as a region with the error-free classification of training samples. What is more, if a given SVM is "certain" about a given input vector, the rest of the models in the ensemble should not be involved in the classification process (to increase classification speed). This led to combining multiple SVMs into a cascade structure, where each node of a such cascade is treated as an expert in its certain regions, that can predict the class of the input vector. One more design goal should be handling the homogeneous regions by early nodes in cascade to further increase classification speed. As building ensembles involves training of multiple base classifiers many of such techniques [31, 57] proposed to use a subset of $\boldsymbol{T}$ in order to train those base models. Compared to those ensemble techniques developed for SVMs the major difference is that the presented solution will optimize $\boldsymbol{T'}$ using the memetic algorithm. This addresses two problems, first, it mitigates the importance of selecting the proper size of the reduced $\boldsymbol{T}$ for each base classifier (as the size is adapted during the computation) secondly, the optimized $\boldsymbol{T'}$ should improve the classification performance compared to the randomly selected subset of $\boldsymbol{T}$.

The high-level flowchart of this new method called CE-SVM—**C**ascades of **E**volutionarily optimized **SVM**s is present in Figure 3.6. Each SVM node within the cascade is designed to specialize in a specific part of the input space whereas, the final SVM node is used to classify examples that do not fit within these certain regions of the space.

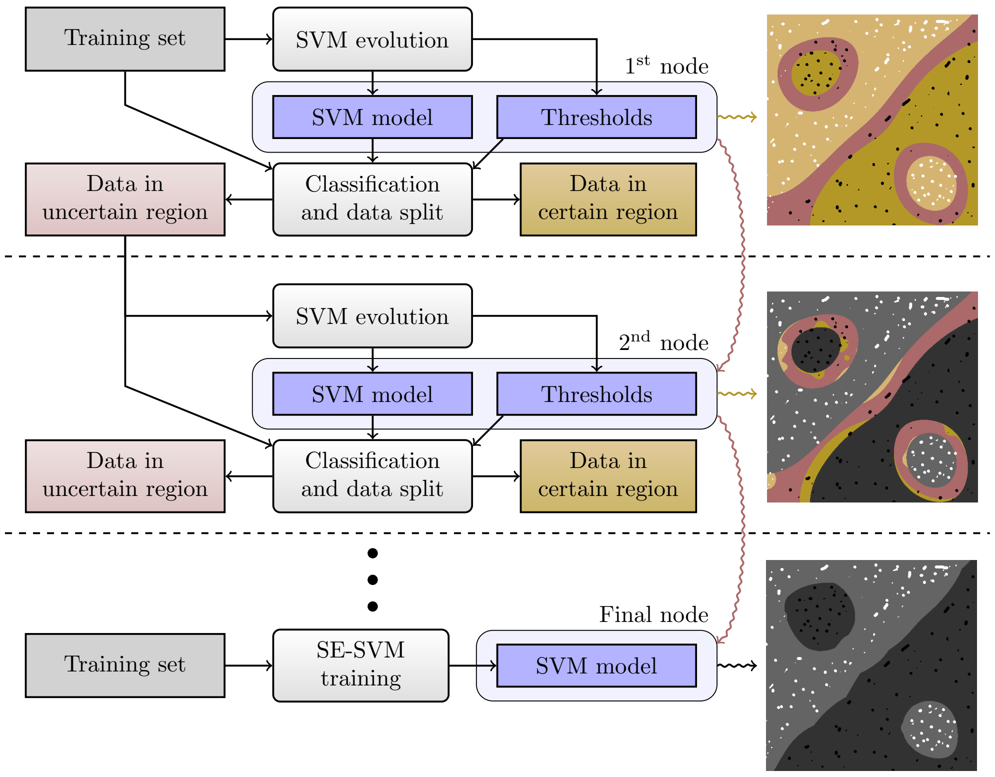


Figure 3.6: Cascade consists of multiple levels, each with a newly evolved node (represented by blue boxes). These nodes are made up of a lightweight SVM model and thresholds that divide the input space into two regions: certain (indicated by yellow shades) and uncertain (indicated by red shade). Any training data that falls into the uncertain region is then passed to the next level, with the final node being based on SE-SVM. The classification process (indicated by waved arrows) is carried out using the evolved cascade. This figure comes from [41].

To divide the dataset into certain and uncertain parts all decisions elaborated by a trained SVM are analyzed (being the distances from the decision hyperplane) and mapped onto the axis after sorting, as presented in Figure 3.7 (these are exemplary values). Individual vectors are represented by vertical black lines (evenly spread for clarity), whereas the green line indicates the SVM decision boundary with zero bias. Certain regions should hold all of the training vectors that are correctly classified. To confront these distances across different SVMs, they are normalized according to the minimal and maximal responses of each model.

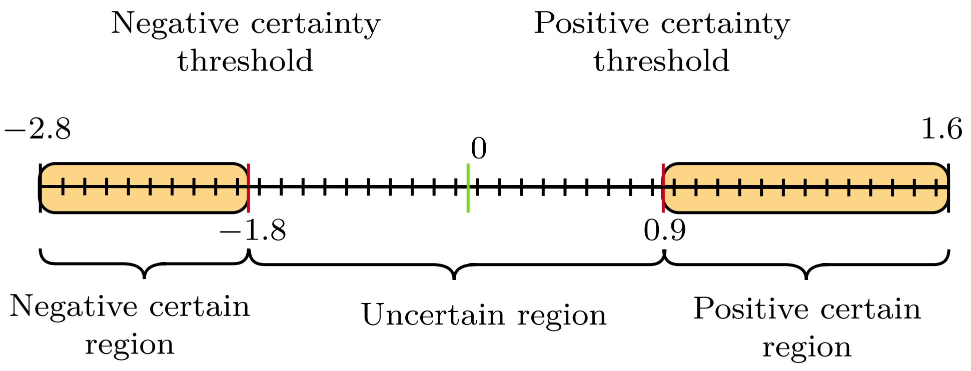


Figure 3.7: Visualization of certainty thresholds. The values presented are only exemplary. This figure comes from [41].

In order to find such thresholds all of the decision values for the data available for training and validation ($\boldsymbol{T}_u \cup \boldsymbol{V}$) are calculated. The $\boldsymbol{T}_u$ represents part of the training set that lay in the uncertain region (at start $\boldsymbol{T} = \boldsymbol{T}_u$). All of those values are sorted in ascending order. Then, starting from the smallest value (the left part of the X-axis in Figure 3.7) the first instance $\boldsymbol{x}_+$ for which the ground-truth class label is positive is found. Next, the average of the decision value obtained for $\boldsymbol{x}_+$ and the decision value obtained for the last correctly-classified instance with the negative ground-truth label becomes the negative certainty threshold. If the threshold is increased beyond $\mathcal{H}_{margin}$, it is cropped to $\mathcal{H}_{margin}$ to incorporate an additional regularization term that assists in avoiding the memorization of training and validation data by overfitting the threshold value to these sets. The positive threshold for the positive certain regions is extracted in a similar fashion.

Building the cascade of SVMs

To construct the cascade of specialized SVMs, it is assumed that all input data ($\boldsymbol{T}$ and $\boldsymbol{V}$) belong to the uncertain region. At each node of the cascade, the training set and hyperparameters of the SVM kernel are evolutionarily optimized using SE-SVM (described previously in Section 3.4). This method was selected based on its time performance of training and classification during initial evaluation and based on the results of our previous papers [38]. As multiple SVMs will be joined here, the speed of training was more important than classification performance (which will be addressed by ensembling the results). When comparing SE-SVM with previous method of DA-SVM it provided much faster training times, and against ALMA it was hypothethised that simultaneous optimization should be better in this case. However, there is one important modification to SE-SVM method, the feature set is not optimized at all and always all of the features are used (the EFS component is disabled). That is due to the fact that each node is modeling part of the input space, but gets SVs from all of the previous nodes. Those SVs are passed down the cascade along with their corresponding γ values and an adaptive RBF kernel is utilized (as in DA-SVM, Section 3.5). This adaptive kernel is used to guarantee the smoothness and continuity of the decision boundary and to provide knowledge to the current node about the previous results in the cascade. The current uncertain region of the input space is reduced using this evolved SVM by classifying all vectors from the previous uncertain region and extracting the thresholds that define the certain region in which the new node specializes. The cascade is iteratively extended with nodes until either the training set is exhausted or it is impossible to enlarge certain regions further by introducing more training and validation vectors using additional SVM nodes. The final node in the cascade is responsible for classifying the uncertain region of the input space.

For the fitness of node evolution, the number of correctly classified samples within a certain region in relation to all of the available samples is used. This fitness encourages selecting the largest possible certain regions. This encourages to select homogeneous regions of input space at early nodes of cascade. One important detail here is that during the calculation of threshold bias term of SVM should not be taken into a certain region. Otherwise, SVM can become "certain" for vectors laying far from all of the SVs where all of the kernel function evaluations are zero (or very close) and the bias term is used to make the decision about the classification result.

As demonstrated in our conference paper [41], the method provides great classification performance. What is more, selected certain regions showed increased accuracy of classification. However, there are several shortcomings with this method after initial tests were conducted. First of all, for some of the sets, the cascade generation was stopped early, thus the coverage of the test set by certain regions was small. This was especially visible in the case of imbalanced sets when the minority class was entirely included into the certain region. Another problem was occurring in the datasets with the high number of features in which certain regions are limited due to data sparsity. Yet another problem was that the "inheritance" of SVs from previous nodes has caused a long training time, especially if the cascade keeps growing (it may also happen when the first few nodes are not covering a large part of the dataset). This led to the development of an improved method presented in the next subsection.

Building ensembles of evolutionary cascades

Firstly, to address the issue of long training times, the adaptive RBF kernel was removed from the design. This modification helped to streamline the overall computational complexity of the architecture, reducing both the training time and computational requirements for each individual node. Additionally, to further enhance the speed and efficiency of the cascade, a random validation set was selected for each cascade (as fitness evaluation can be longer than SVM training). In all previous methods, the $\boldsymbol{V}$ was based on the dataset split done before running the algorithm. It often happened that, for large datasets, the evaluation of $\boldsymbol{V}$ took more time than training the SVM classifier (especially at the beginning of evolution where K_t was relatively small). So selection of random and small $\boldsymbol{V}$ should increase the speed of building a cascade, and help to focus training on specific regions of the input space. What is more, these random $\boldsymbol{V}'s$ should increase the diversity among the cascades as they would fit to different distributions of data (which should result in better classification) [113]. As previously mentioned, joining different kernels could be beneficial. In this case, two separate populations were evolved simultaneously, one with a linear SVM and the other with an RBF kernel. It means that nodes in a single cascade could have different kernels used. This approach allowed for greater flexibility in the design of the cascade, enabling it to adapt to a wider range of input data. Finally, to further improve the classification performance of the system, the uncertain region of the input space was handled by an extra tree (ET) classifier (a type of a random forest). Combining multiple different base models can further improve the classification performance and is often more successful than homogeneous ensembles (the ones built with base models of the same type of classifier—SVMs) [111]. Please note that ET is utilized only for handling uncertain regions and is not used otherwise. All of those modifications should improve the shortcomings of CE-SVM method and prove the second research hypothesis that building ensembles of SVM using evolutionary algorithms provides improved classification performance.

Constructing multiple cascades allowed for the trade-off between classification performance and computational efficiency of a single cascade, wherein the optimization of the latter was prioritized. Compared to the CE-SVM method, a single cascade might provide worse classification quality. However, even when one of the cascades fails to cover a large part of the input space and/or provides poor results, it can be improved and mitigated by other ones.

Algorithm 9 Code for building ensembles of evolutionary SVM cascades.

```
 1: $E \leftarrow \emptyset$, $\epsilon \leftarrow 0.001$
 2: repeat
 3:     $\boldsymbol{T}$, $\boldsymbol{V} \leftarrow$ Resample($\boldsymbol{T} \cup \boldsymbol{V}$)
 4:     $\varepsilon_i \leftarrow$ BuildCascade($\boldsymbol{T}$, $\boldsymbol{V}$)
 5:     $\varsigma_i \leftarrow$ ScoreLevelWise($\varepsilon_i$, $\boldsymbol{T} \cup \boldsymbol{V}$)
 6:     $MCC_\Delta$, $\boldsymbol{T}_u$, $\boldsymbol{V}_u \leftarrow$ Evaluate($E$, $\varepsilon_i$,$\varsigma_i$, $\boldsymbol{T} \cup \boldsymbol{V}$)
 7:     $Unceratin_\Delta = |\boldsymbol{T}_u|/|\boldsymbol{T}| + |\boldsymbol{V}_u|/|\boldsymbol{V}|$
 8:     $E \leftarrow E + (\varepsilon_i, \varsigma_i)$
 9: until $MCC_\Delta > \epsilon$ or $Unceratin_\Delta > 1\%$
10: return $E$

11: function BuildCascade($\boldsymbol{T}$, $\boldsymbol{V}$)
12:     $\varepsilon \leftarrow \emptyset$, $\boldsymbol{T}_u$ $\boldsymbol{V}_u \leftarrow \boldsymbol{T}$ $\boldsymbol{V}$
13:     repeat
14:         $SVM_{\text{Line}} \leftarrow Evolve(\boldsymbol{T}_u, \boldsymbol{V}_u, Linear)$
15:         $SVM_{\text{RBF}} \leftarrow Evolve(\boldsymbol{T}_u, \boldsymbol{V}_u, RBF)$
16:         $SVM \leftarrow \max(\eta(SVM_{\text{Line}}), \eta(SVM_{\text{RBF}}))$
17:         $\varepsilon \leftarrow \varepsilon + SVM$
18:         $\boldsymbol{T}_u, r_T \leftarrow$ GetUncertain($SVM$, $\boldsymbol{T}_u$)
19:         $\boldsymbol{V}_u, r_V \leftarrow$ GetUncertain($SVM$, $\boldsymbol{V}_u$)
20:     until $r_T$ or $r_V$ or $|\boldsymbol{T}_u| > K_t$
21:     return $\varepsilon$
```

The process of building such ensemble is presented in Algorithm 9. The first line begins, with initialization of an empty ensemble and ϵ value for improvement in MCC score required by new cascade. The MCC score was used as AUC is difficult to calculate and interpret for a cascade classifier. After that, the process of building ensemble begins. At the beginning (line 3), the training and validation sets are joined and resampled, where up to ϑ vectors are selected for $\boldsymbol{V}$ and no more than ζ of all data. This division is done in a random manner (keeping the class stratification of the dataset) and passed to process of building cascade (line 4). During this process, all of the training and validation data is considered as uncertain region at first (line 12). Then the nodes are trained using a SE-SVM algorithm, but without feature selection process, the same as in the CE-SVM method (based on the same observations). The fitness function for evolving single node within cascade is also taken from CE-SVM and is based on the certain region coverage in relation to available data. Two populations are trained independently using different kernels, namely linear and RBF (lines 14-15). After the training, the model with a higher fitness score is selected (line 16) and appended to the cascade. Then this model is used to find new uncertain regions of the space (shrinking current $\boldsymbol{T}_u$ and $\boldsymbol{V}_u$). During that process, there is an additional check whether the size of the given uncertain set was reduced (r_T for $\boldsymbol{T}_u$ and r_V for $\boldsymbol{V}_u$). The cascade is being built until there are no new vectors added to certain region or the training data have sufficient size allowing to build new nodes. Otherwise, the evolution of the cascade is finished and it is returned to the main algorithm that builds ensemble (line 21). Later in line 5, this cascade is scored where each node receives MCC score in its certain region. Those scores are used during the classification process which uses voting scheme and these MCC scores act as weights. Thanks to that if just a single node has poor performance it is known and its answer could be easily discarded (or outvoted). Then the new cascade is evaluated with the whole ensemble on the joined training and validation set. Here the improvement of MCC score over the whole ensemble (MCC_Δ) is calculated and uncertain sets are selected (line 6). During the evaluation, at least half of the cascades present in the ensemble need to provide an answer within the certain regions to consider this answer as a certain one. Otherwise, such a vector is classified as lying in the uncertain region. In next line 7, the size of the uncertain set with this new cascade is calculated and the cascade is added to the ensemble (line 8). The stop condition, when new cascades should not be added into the ensemble is when MCC score of the whole ensemble did not improve (by at least ϵ) when adding a new cascade, or the uncertain region did not decrease in size by more than Υ of the $\boldsymbol{T} \cup \boldsymbol{V}$. When one of that conditions is fulfilled the process ends and the ensemble is ready.

3.7 Summary

All of the presented methods are summarized by their main features in Table 3.3. Please note that the first two techniques (GASVM [75] and MASVM [96]) are earlier works by Nalepa and Kawulok. While the presented work uses both of those methods, it extends them significantly by combining them with hyperparameter optimization and feature selection. Compared to Table 3.1, this summary provides a timeline of the publications and highlights the difference between them. The GASVM and MASVM focused only on providing training set selection for SVM. Their results showed that training SVM with $\boldsymbol{T'}$ is viable for resulting classification performance and decreasing computational costs. However, the main disadvantage of those methods was the need of specifying the SVM hyperparameters beforehand. As presented earlier this task is not trivial and SVM performance depends on it. That was the initial motivation for proposing the ALGA and ALMA methods, where hyperparameter optimization is coupled with training set selection. This method formed a framework where two separate optimization processes are run. These processes are joined in an alternating scheme, where the result obtained by one of those processes is used during other optimization (the result of hyperparameters optimization is used in the training set selecting and vice-versa).
This alternating scheme was later extended by incorporating a feature selection process. The FSALMA used a preselection technique, the RFECV algorithm to optimize the features set before running optimization whereas, the E-SVM method incorporated the presented EFS for feature selection. However, it was hypothesized that the alternating scheme might be suboptimal, hence the simultaneous evolution of hyperparameters, training set, and feature set was introduced with SE-SVM method. It should provide a better classification quality compared to previous methods, while training and classification times should be improved compared to algorithms not performing training set selection. However, SE-SVM is not expected to be faster than GASVM or MASVM as it is a more complex method for optimizing also hyperparameters and feature sets. These additional optimization processes are expected to negatively impact the run-time of the method. Although

Table 3.3: The most important aspects of SVMs optimized by evolutionary techniques.

Method	Year	***F***	**Model**			***T***	**Uncertainty**	**Ensemble**
			Single kernel	Var. γ in RBF	Mixture			
GASVM [75]	2012	✗	✗	✗	✗	✓	✗	✗
MASVM [96]	2014	✗	✗	✗	✗	✓	✗	✗
ALGA [76]	2017	✗	✓	✗	✗	✓	✗	✗
ALMA [39]	2018	✗	✓	✗	✗	✓	✗	✗
FSALMA [39]	2018	✓	✓	✗	✗	✓	✗	✗
E-SVM [37]	2019	✓	✓	✗	✗	✓	✗	✗
SE-SVM [38]	2019	✓	✓	✗	✗	✓	✗	✗
ARBF-SVM [94]	2020	✗	✗	✓	✗	✓	✗	✗
DA-SVM [40]	2021	✓	✗	✓	✓	✓	✗	✗
CE-SVM [41]	2022	✗	✗	✓	✗	✓	✓	✓
ECE-SVM	-	✗	✗	✗	✓*	✓	✓	✓

GASVM—**G**enetic **A**lgorithm for selecting **SVM** training sets.
MASVM—**M**emetic **A**lgorithm for selecting **SVM** training sets.
ALGA—**AL**ternating **G**enetic **A**lgorithm for selecting **SVM** training sets and models.
ALMA—**AL**ternating **M**emetic **A**lgorithm for selecting **SVM** training sets and models.
FSALMA—**AL**ternating **M**emetic **A**lgorithm for selecting **SVM** training sets and models with **F**eature **S**election.
SE-SVM—**S**imultaneously-**E**volved **SVMs**.
E-SVM—**E**volutionarily-tuned **SVMs**.
ARBF-SVM—**SVMs** with **A**daptive **RBF** kernels.
DA-SVM— **D**ata-**A**daptive **S**upport **V**ector **M**achines.
CE-SVM—**C**ascades of **E**volutionarily optimized **SVM**s.
ECE-SVM—**E**nsembles of **C**ascades of **E**volutionarily optimized **SVM**s.
* the mixture of kernels in ECE-SVM is not using the same mixing as in DA-SVM algorithm.
In ECE-SVM different kernels are present in separate nodes not in single SVM.

compared with the fact that GASVM and MASVM required GS to provide the proper hyperparameters for SVM, the total run-time should be reduced (which is directly related to the first research hypothesis stated in Section 1.1). During the experimentation with those methods, it was observed how the γ hyperparameter (of RBF kernel) affects the classification performance. Thus, two new algorithms using adaptive RBF kernel were introduced. They offer a Pareto-optimal solution compared to SE-SVM, where those methods should provide better classification performance by trading off the training and classification time. To further extend the performance of SVMs models, an ensemble scheme was proposed. It focused on finding so-called certain regions where classification quality should be increased (as opposed to uncertain regions). This division of the input space allowed the building of a cascade structure of SVMs, where each node is an expert in its region. This technique was introduced with CE-SVM method. The experiments in [41] unveiled a few shortcomings of this method, where building such an ensemble was stopped too early. Moreover, the training times were quite long in the case of building a cascade with many nodes, where the process was getting slower with each new node. This led to the development of the final method ECE-SVM which addresses these issues. It should provide the best results in terms of classification quality while training and classification times should be on par with CE-SVM method. The last of the presented algorithms ECE-SVM is not yet published but it is planned to be submitted to a journal.

Chapter 4

Experimental validation

This chapter shows the results of experimental validation where the most important algorithms introduced in this dissertation are tested. Their performance is investigated using two types of datasets: (1) artificially generated 2D datasets and (2) multiple benchmark datasets taken from popular repositories.

The chapter is organized as follows: first, the datasets are briefly described and presented (Section 4.1) with reasoning why they were selected for this study. Next in Section 4.1 the experimental setup and implementation details are presented. Following that, in Section 4.2.1 each algorithm is validated independently using 2D datasets and all of them are compared using those artificially generated datasets. In Section 4.2.3 those proposed approaches are compared to each other and compared to other popular classifiers and state-of-the-art methods using benchmark datasets.

4.1 Datasets

The effectiveness of the proposed algorithms will be tested using two kinds of datasets:

- **Artificially generated datasets**—they include vectors that are either manually created or produced to adhere to a predetermined distribution, such as the Gaussian distribution. As a result, the underlying characteristics of the data are known, which is not always the case with benchmark and real-world datasets. Additionally, artificially generated datasets are often simple to visualize. These datasets are used to study the behavior of new algorithms. This study includes several artificially generated datasets (see examples in Figure 4.1, where white and black pixels represent vectors from the positive and negative classes, respectively).

- Benchmark datasets—datasets of this type have various characteristics and are used to evaluate the effectiveness of algorithms. They ease comparison among the methods and provide points of reference as well as prove the usefulness of the methods as these types of datasets are often based on real-life data. There are three popular repositories from which datasets were gathered for this study:

  1. UC Irvine (UCI) machine learning repository: `https://archive.ics.uci.edu/ml/index.php5`
  2. Knowledge Extraction based on Evolutionary Learning (KEEL) repository: `http://www.keel.es/`
  3. LibSVM repository: `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`

  In Table 4.2, benchmark datasets used are denoted with the name of the repository where they came from. Although the sizes of most of these benchmark datasets are not very large, they are widely used in the literature to compare different algorithms that optimize various aspects of SVMs.

All of the presented datasets concern binary problems. Even though SVMs could be extended to multiclass classification problems by employing the one-vs-one or one-vs-all scheme, this kind of solution might be suboptimal as presented in [40]. The algorithms discussed in the previous section could be used with specialized methods that handle multiclass classification as there is no inherent limitation to the number of classes. However, this is beyond the scope of this dissertation.

All of the artificially generated datasets are presented in Table 4.1. The imbalance ratio (IR) is reported for each dataset, it is calculated as a ratio of the more numerous class to the less numerous class (so IR ≥ 1). The datasets were created by "painting" a 2D canvas with black and white vectors as presented in

Table 4.1: The artificial created 2D datasets use in the analysis. IR stands for imbalance ratio, $\sum$ is the number of vectors in the dataset before any divisions where $\boldsymbol{T}$ stands for the number of vectors in the training set and Ψ is the number of vectors in the test set.

	IR	$\sum$	$\boldsymbol{T}$	Ψ
2D-Shapes	1.61	6458	1160	5298
2D-Blobs	1.26	34957	6371	16767
2D-Dots	1.42	2842	242	1552
2D-Chessboard	1.29	69971	19949	22169
2D-Circles noise	1.51	60462	36278	12092
2D-Grain blobs	1.95	46555	27933	9311
2D-Imbalanced line	2.00	26500	15900	5300
2D Line	1.25	13228	3600	5606
2D-Line wit Dot	1.64	55165	33099	11033

	IR	Sum	$\boldsymbol{T}$	Ψ
2D-Dense	1.10	102301	61381	20460
2D-Linear pool	1.11	18666	4943	6767
2D-Linear pool flip	1.11	18666	4943	6767
2D-Many pools	1.24	37110	22266	7422
2D-Mixed	1.18	34434	20662	6886
2D-Mixed heavy	4.07	19084	11452	3816
2D-Stripes	1.60	40237	24143	8047
2D-X	1.45	80506	48304	16101

Figure 4.1, where all test folds (Ψ) are visualized. Each dataset contains a single division into training, validation, and test data. These datasets were created in order to better understand the behavior and performance of the algorithm during the design phase. Thus, each of such datasets has its own unique character, which will be briefly described.

The `2D-Shapes` dataset present a wiggled line with a small margin between them where the test set is much bigger than the training set (Figure 4.1a). The `2D-Blobs` dataset (Figure 4.1b) was created to test patterns of different densities with large empty spaces, contrary to that `2D-Dots` show four sparse patterns in each corner of the image which have wide margins between them. The chessboard pattern seen in Figure 4.1d introduced a high number of samples and difficult-to-model decision boundary with varying densities (see bottom right corner). The circles with noise (Figure 4.1e) are an interesting pattern of concentric circles that belongs to different classes. Its purpose was to test if the selected SVs will also create such a pattern and be evenly distributed. The `2D-Grain blobs` (Figure 4.1f) test the behavior of non-linear data that has a sparse representation but covers most of the input space without much of the "empty" space without any vectors. Next datasets in Figures 4.1g - 4.1l present different variants of data that have a clear linear division (Figure 4.1g) with the intrusion of non-linearly separable data (the rest of examples). Those datasets were intended to test the selection process of the training set as well as help to test adaptive kernel and ensemble building schemes—whether the "pools" of non-linear data will be handled first or it does not matter. As it can be seen, most datasets provide quite a clear margin between classes, so algorithms were expected to get results close to perfect classification even on the test set. Nonetheless, a lot of those datasets helped to catch problems at the early stages of development and were a good benchmark for the tested solution. The last two datasets (Figure 4.1n and 4.1o) were introduced to test how well the algorithm would work if there was no clear and sharp boundary between data points and if the resulting model did not overfit. Here the expected scores should be far from perfect (meaning no mistakes on Ψ). Please note that these datasets were added during the development of subsequent algorithms so not all of them were created at the beginning for the ALMA method (the first algorithm chronologically). What is more with the ability to control the size of the sets they ranged from quite small (as in the case of `2D-points`) to what could already be considered as a large dataset for SVM with over 100 000 samples. Furthermore, most of the datasets were kept balanced.

The benchmark datasets were selected to account for many different scenarios with diverse imbalance ratios as well as the dataset sizes and number of features. As training set selection is one of the vital points in the presented algorithms, very small datasets were excluded from the study. In the case where the number of samples is small, it would be better to train a classifier on all of the available data or even use a dedicated method to generate new samples (e.g., Synthetic Minority Oversampling Technique (SMOTE) algorithm [25]). In the case of UCI sets, a five-fold cross-validation approach is employed with stratification to ensure that each fold contains an equal ratio of vectors from both classes. Furthermore the $\boldsymbol{T}$ was divided into four non-overlapping parts and one of them is used as the $\boldsymbol{V}$. Hence there are five stratified folds divided into $\boldsymbol{T}$, $\boldsymbol{V}$ and Ψ with a 3:1:1 proportion. On the other hand, in KEEL, the sets are initially divided into five non-overlapping folds with distinct training ($\boldsymbol{T}$) and test (Ψ) sets and are kept without any changes. The datasets taken from LibSVM repository were also divided into the distinct training ($\boldsymbol{T}$) and test (Ψ) sets in all cases. Hence, there is no separate $\boldsymbol{V}$ for both the KEEL and LibSVM datasets—$\boldsymbol{T} = \boldsymbol{V}$ was used in that case, as the validation set is needed during the evolution process of presented algorithms. There is no risk in overfitting the SVM classifier, as thanks to training set selection the final training set used for SVM classifier is different from the validation set $\boldsymbol{T'} \neq \boldsymbol{V}$.

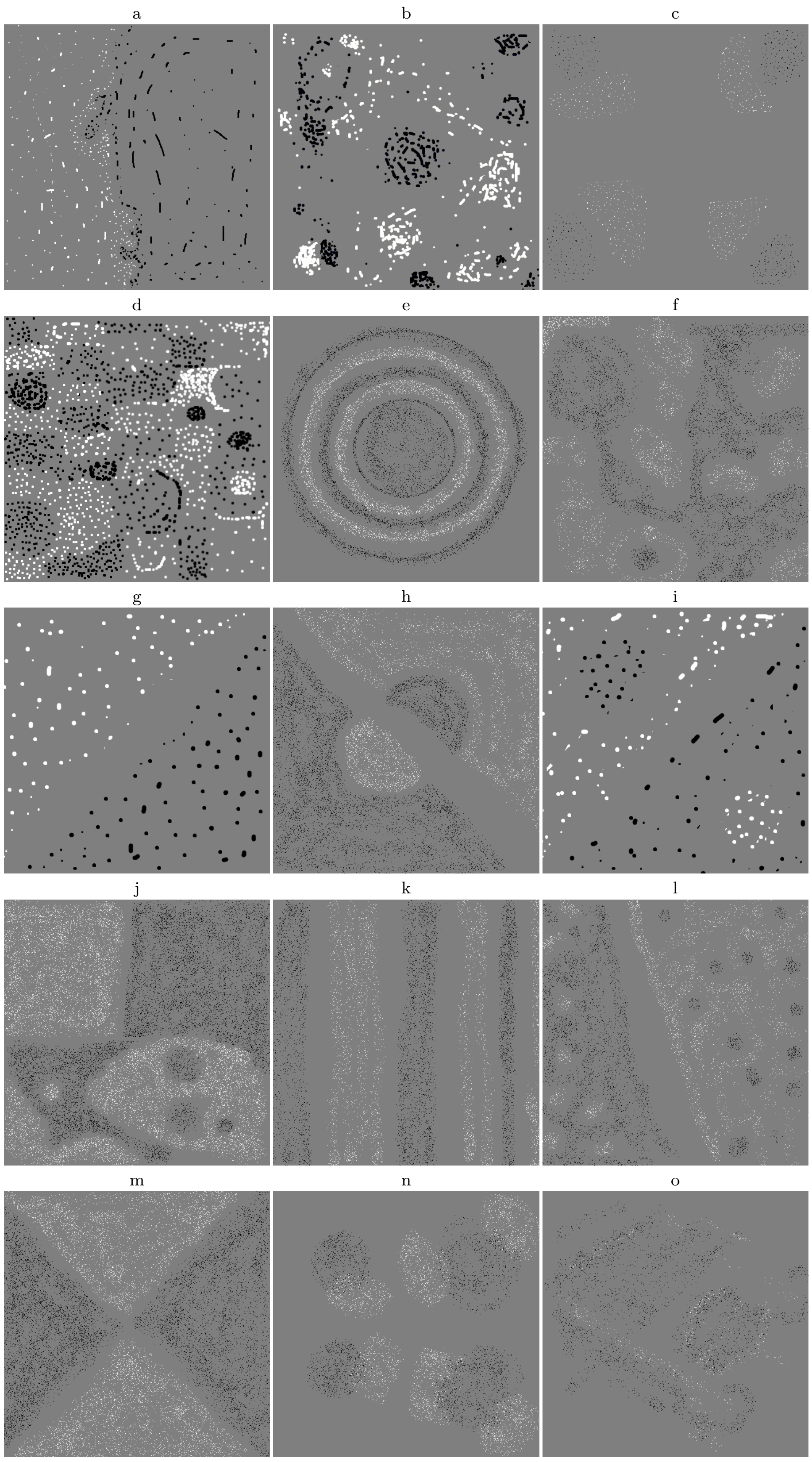


Figure 4.1: Visualization of tests sets for 2D datasets.

Table 4.2: The benchmark datasets used in the analysis.

Dataset	IR	$\sum$	$\boldsymbol{T}$	Ψ	$\boldsymbol{F}$
a1a	3.15	32561	1605	30956	123
a2a	3.15	32561	2265	30296	123
a3a	3.15	32561	3185	29376	123
a4a	3.15	32561	4781	27780	123
a5a	3.15	32561	6414	26147	123
a6a	3.15	32561	11220	21341	123
a7a	3.15	32561	16100	16461	123
a8a	3.15	32561	22696	9865	123
a9a	3.18	48842	32561	16281	123
australian	1.25	690	551	139	14
banana	1.23	5300	4239	1061	2
banknote	1.25	1372	1097	275	4
cod	2.00	488565	331152	157413	8
coil2000	15.76	9822	7856	1966	85
covtype	1.05	581012	464809	116203	54
credit card clients	3.52	30000	23999	6001	23
german	2.33	1000	800	200	24
magic	1.84	19020	15215	3805	10
mnist	1.03	70000	60000	10000	784
page-blocks	8.79	5472	4377	1095	10
phoneme	2.41	5404	4322	1082	5
pima	1.87	768	614	154	8
ring	1.02	7400	5919	1481	20
segment	6.02	2308	1846	462	19
skin	3.82	245057	196045	49012	3
spambase	1.54	4597	3662	935	57
twonorm	1.00	7400	5919	1481	20
vowel	9.98	988	790	198	13
wdbc	1.68	569	454	115	30
winequality	1.73	6497	5197	1300	11
wisconsin breast	1.86	683	546	137	9

All datasets were normalized by rescaling the features to $[0, 1]$ [65]. The rescaling was done on the training set so the actual values in the test set might be out of $[0, 1]$ range. This is important as no information from the test set should be leaked in the process of learning the classifier.

Experimental setup

The algorithms were implemented in `C++` (Visual Studio Community 2017) with the LibSVM library [24], and the experiments ran on the machine equipped with the Intel i9-7900X CPU and 64 GB of RAM under Windows 10 operating system. For other classifiers described, we exploited the Scikit-learn (version 0.20.2) implementation in Python 3.6 if not stated otherwise. Hyperparameters of presented methods follow settings proposed in previous publications.

4.2 Results

This section presents all of the results and analysis of the proposed methods. First, it will start with 2D visualizations of algorithms, their main principles and a qualitative comparison over selected examples of the 2D datasets. Furthermore, the quantitative analysis will be performed over artificial datasets to select the best methods for final comparison among other algorithms and state-of-the-art methods for SVM optimization.

Not all algorithms will be tested in the following sections. First of all, ALGA method had a major drawback in the need to select proper K_t value in order to achieve good performance which requires checking multiple values (this is time costly process). However, this drawback was removed in the later work with MASVM [96] and ALMA methods, which will be used as the baseline techniques. Moreover, FSALMA and E-SVM introduced a feature selection mechanism that is not useful in the case of 2D artificial datasets as always all of the features are selected. Besides that, both FSALMA and E-SVM are surpassed by the SE-SVM algorithm which presented superior results to both of those methods in our paper [38]. Furthermore, feature selection is an interesting extension of the presented methods but needs to be studied in more detail. Regarding the DA-SVM method, only the best variants presented in [40] have been investigated here (in order to save computation time). For the building of ensembles, both CE-SVM and ECE-SVM methods have been tested.

4.2.1 Qualitative analysis

First, the process of evolution of the algorithms is visualized and discussed based on 2D datasets. The algorithms are presented in the same order as they were introduced in Section 3. The visualizations in question do not aim to present every single generation that occurs during evolutionary computation. Rather, their purpose is to highlight specific instances where the current best solution has changed, indicating an interesting development that can help to understand how the algorithm works. By identifying these key points in evolution process, we can gain a deeper understanding of the underlying dynamics of proposed methods. What is more, the best solution found by EAs does not need to change in each generation (based on how the proposed methods work).

The evolution process of ALMA is presented in Figure 4.2. First, in Figure 4.2a, a random subset of vectors is selected for the training set alongside default parameters. This pair is used to train the presented SVM classifier. This initial solution offers a quite poor solution to the problem. In Figure 4.2b, the optimization process of hyperparameters starts. The training set remains unchanged throughout this phase which can be seen, as the same SVs are present in Figure 4.2a and 4.2b. In

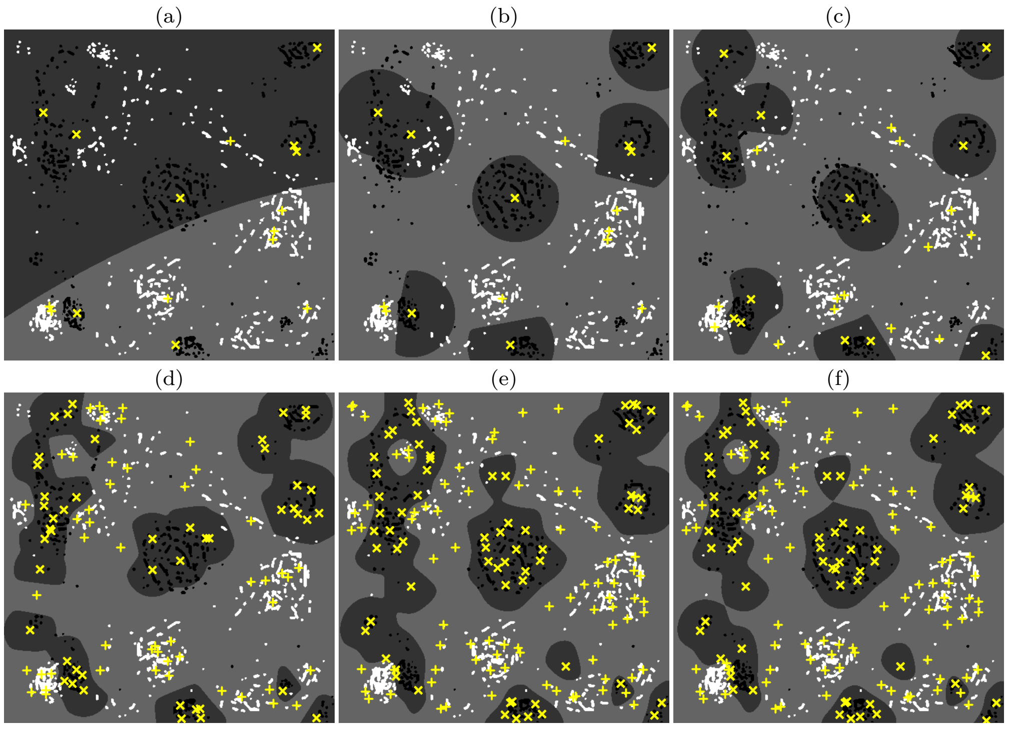


Figure 4.2: Visualization of the ALMA algorithm run. Results are visualized on 2D-Blobs test set. The yellow ticks and crosses mark the positions of support vectors: (a) presents a solution after initialization, (b) presents the end solution after kernel evolution phase, (c)—(e) shows progress of $\boldsymbol{T'}$ evolution and growing of its size (f) presents final solution where hyperparameters were adjusted.

later stages of evolution, this might not be so obvious as when $\boldsymbol{T'}$ grows not all of the training vectors might become SVs. The process of changing hyperparameters values (during optimization) affects which vectors become SVs. When no more improvements could be made optimization switches to the training set selection in Figure 4.2c. What is crucial in this algorithm (and this phase) is the fact that the size of $\boldsymbol{T'}$ can grow as seen in Figure 4.2d and Figure 4.2e. At the beginning of the optimization process phases may change more rapidly, where each phase lasts just for a few generations. Often a pivotal point of the algorithm is when the training set size grows multiple times in a single phase, as this means that the values of the current hyperparameters are well selected. In the last stages the overall improvements are rather minor as seen in Figure 4.2e and 4.2f, where an adjustment to hyperparameters is made (the γ value gets larger between those two). Finally when no more improvements could be made the process stops and the best solution is returned (as seen in Figure 4.2f).

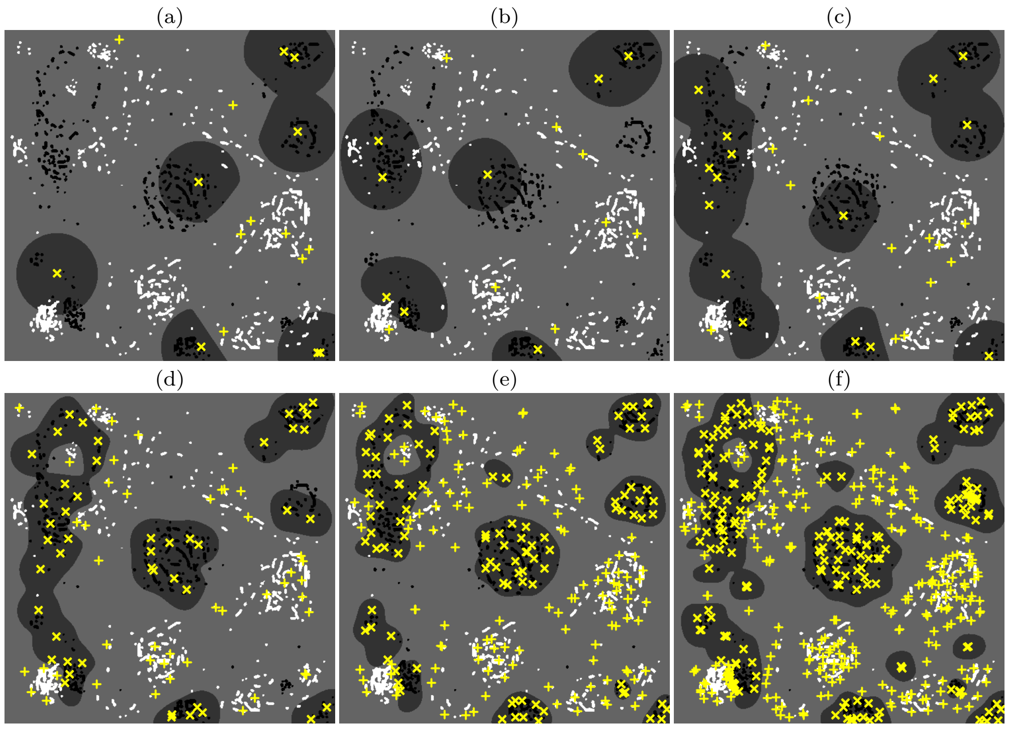


Figure 4.3: Visualization of the SE-SVM algorithm run. Results are visualized on 2D-Blobs test set. The yellow ticks and crosses mark the positions of support vectors: (a) presents a solution after initialization, (b) presents the improved solution where $\boldsymbol{T'}$ did not grow yet, (c)—(e) shows the progress of $\boldsymbol{T'}$ growing and hyperparameter adjustments (f) presents the final solution.

Contrary to the ALMA process, SE-SVM can change both the $\boldsymbol{T'}$ and hyperparameters at the same time. The first advantage of this approach is visible by comparing Figure 4.2a and Figure 4.3a, where in the case of SE-SVM initialization provides better modeling of input space. Here initially the γ values tend to be lower (have a larger range of influence), as the training set size is small, and can adapt more dynamically when it grows. This is visible comparing Figure 4.3a to Figure 4.3d. This algorithm proves to provide better results as the mutual dependence between the training set and its "optimal" hyperparameters is visible and used as an advantage. In this particular case, SE-SVM also shows the ability to change the dynamics of adjustment for γ. Between Figures 4.3e to Figure 4.3f γ value is lowered while in all other examples (a-d), it was getting bigger (lowering the range of influence).

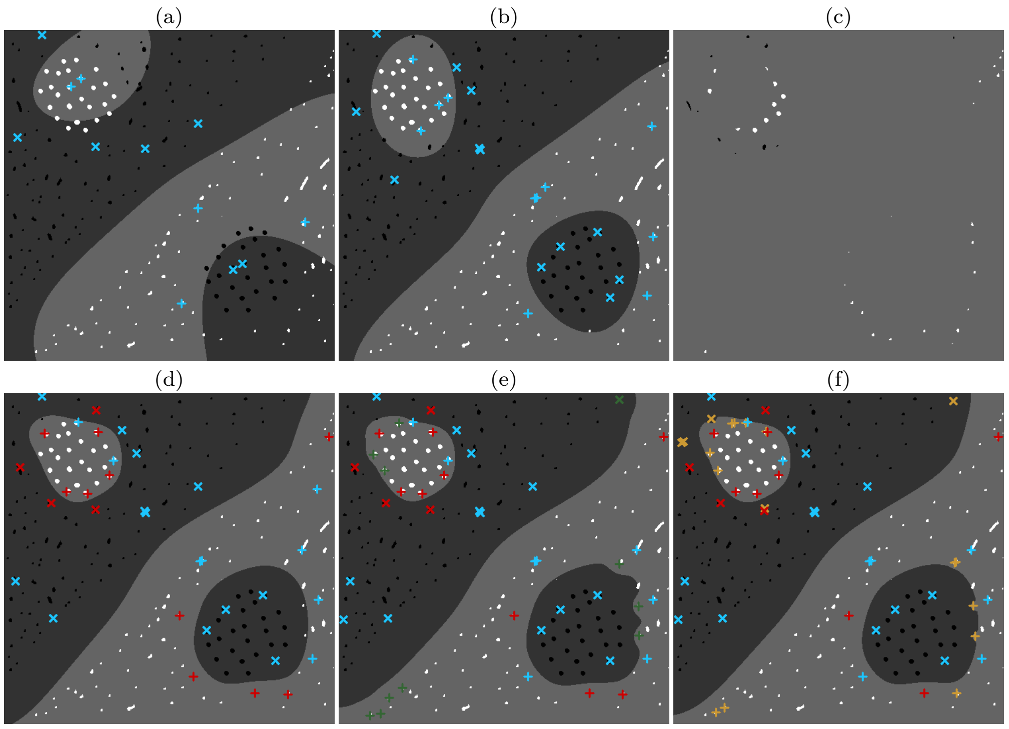


Figure 4.4: Visualization of the ARBF-SVM algorithm: (a) presents the best solution from the initial population, (b) The best solution after finishing evolution with first γ from $\vec{\gamma}$, (c) Shrank training set that will be used in next iteration with subsequent γ. The shrinking procedure is based on the whole population, (d) Solution after second evolution has ended. Added new support vectors marked with red color crosses, (e) Adding next γ value marked with green vectors provided worse classification performance, these support vectors will be removed, (f) final solution for a given dataset containing three different γ values.

The DA-SVM (and ARBF-SVM) works in a different manner than previously shown examples. First of all, it uses an adaptive kernel so vectors with different γ's are denoted with colors, where each color presents a unique value of γ. In Figure 4.4a, the best solution from the initial population is shown which is then evolved. This evolution process ends, and yields the solution visible in Figure 4.4b. Now according to this solution, the original $\boldsymbol{T}$ is being shrunk so only vectors visible in Figure 4.4c will be used in subsequent optimization that will use larger γ value.

The result of the next round of iteration is presented in Figure 4.4d. In Figure4.4e the next solution with new green vectors is present. However, it provides worse classification performance so these results are discarded (SVs with those γ values are not added to the trained SVM model). This means that the selected γ had a too-low value and added too much fluctuation into the decision hyperplane. Please note that when results are discarded, the shrunk $\boldsymbol{T}$ used for the next iteration is the same as the one obtained after the solution presented in Figure 4.4d. Finally, the resulting solution is presented in Figure 4.4f, where three distinctive γ values are present. This process was presented on a different dataset than ALMA and SE-SVM as in previous examples the process of discarding certain γ did not occur. This process was not present, because `2D-Blobs` datasets present more variability in the terms of heterogeneous and homogeneous regions as well as their sparsity so each time adding new γ helped to improve the classification performance and better model the data.

The evolution process of CE-SVM uses SE-SVM algorithm as its base for building the nodes of the cascades, although the process of feature selection is disabled (only $\mathcal{M}$ and $\boldsymbol{T'}$ are being optimized). First of all, feature selection does not make sense on the 2D datasets. Moreover, the structure of the cascade made it difficult to apply the feature selection process on each node, as the next node in the cascade depends on the previous one by using the adaptive RBF kernel. In Figure 4.5a, the beginning of the process of building cascade is presented. For clarity, SVs are not visualized. The dark and light gray areas present certain regions of previous nodes, the yellow colors newly added (by the next node in the cascade) certain regions, and red regions represent uncertain regions that are not yet covered by the cascade. As presented in the first row (Figure 4.5a-c), the evolution of such a cascade adds new certain regions, which could have different coverage of each class and could be imbalanced. Note that in this solution previously selected SVs are passed into the lower levels of the cascade to make the next node aware of previously obtained classification. This provides a much “smoother” decision hyperplane and enables the extension of some regions by a small extent as seen in Figure 4.5b. Moreover, all vectors from the certain region are removed for the building process of a new node in the cascade (from both the training and validation set). The final cascade is visualized in Figure 4.5d. What could be appreciated is that the

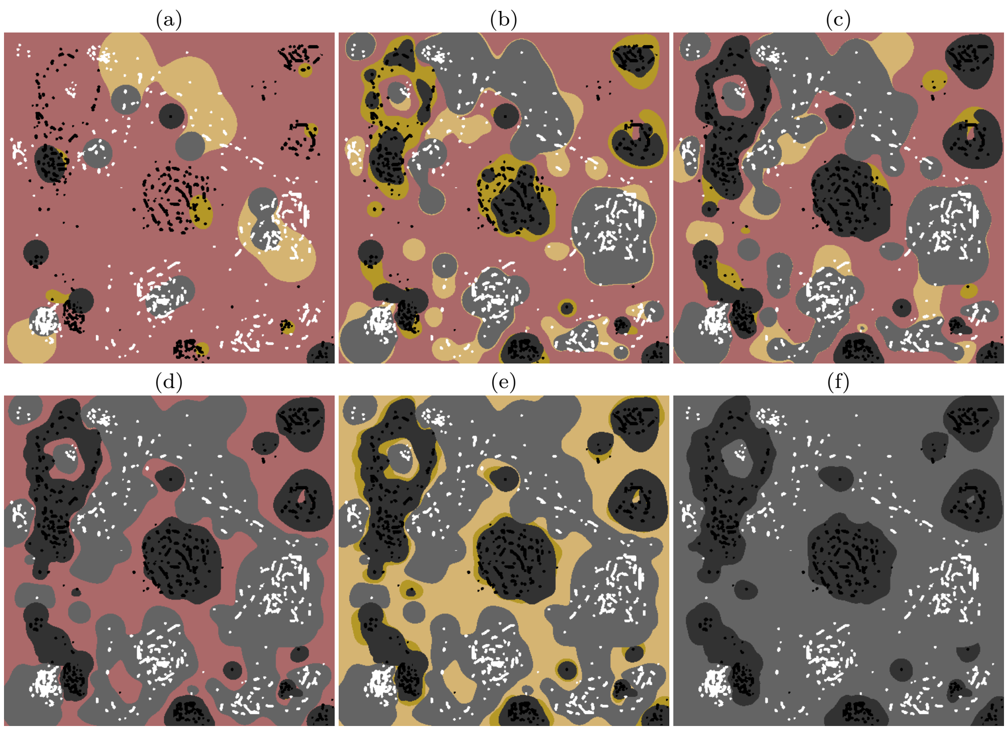


Figure 4.5: Visualization of the CE-SVM algorithm run: (a)-(c) show the evolution process and adding new nodes. Red color denotes uncertain region, dark and light grey presents certain regions of previous nodes while yellow colors are used to depict new certain regions added by the current node. (d) shows final classification with uncertain regions (e) presents how SE-SVM node to resolve those regions works and (f) shows the final classification result of joining cascade with SE-SVM.

majority of the uncertain regions (red color) do not have any vectors in them. For this region, the answer of such a cascade would tell that the sample could not be classified (the "unknown" class). In order to compare this algorithm with other solutions, an SE-SVM is run to cover all of the uncertain parts of the dataset. The behavior of this final node is presented in Figure 4.5e, where previously uncertain regions are shown to belong to positive (light yellow) or negative class (gold). The resulting classification of such a combination of cascade with the SE-SVM to solve uncertainty is presented in Figure 4.5f.

The last algorithm of ECE-SVM is presented in Figure 4.6. It works by building a list of cascades, giving them weights based on their performance on $\boldsymbol{V}$ (each node in cascade has its separate score), and employing a weighted voting strategy

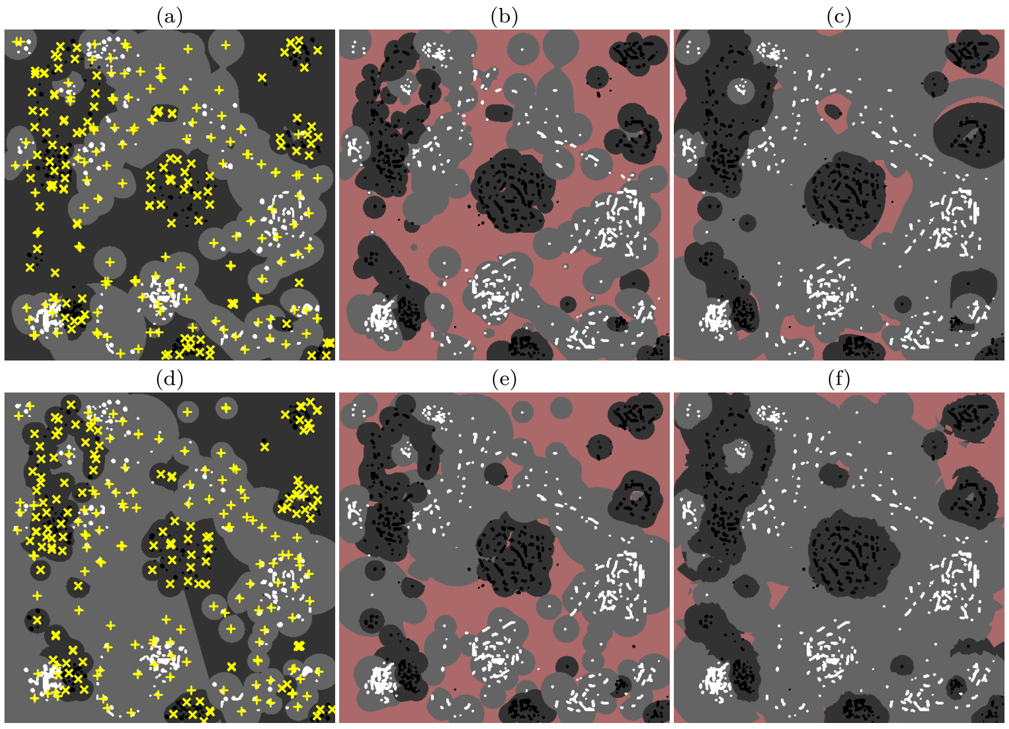


Figure 4.6: Visualization of the ECE-SVM algorithm run: (a) and (d) present a single cascade result, while (b) and (e) show the same cascades but with uncertain regions depicted in red, (c) presents another example of a different cascade. (f) presents the final combination of all of the cascades.

to combine the results of individual cascades. What is more, only cascades which respond within a certain region are considered and at least half of the cascades in the ensemble need to provide a valid response. Otherwise, the region is considered as uncertain. In Figure 4.6a and 4.6d, the example of cascades with denoted all of their SVs are presented. In Figure 4.6b-c and 4.6e, there are examples of the cascades with uncertain regions present. Compared to CE-SVM, the process of building a single cascade is changed. The SVs are not inherited and there is a much smaller validation set selected among other changes (more details are provided in Section 3.6). The main reason behind those changes is to speed up the building process of the single cascade, while its classification performance could be worse. This is covered by the usage of multiple cascades that form the final ensemble which is presented in Figure 4.6f. Although the uncertain region is smaller than the one of CE-SVM (Figure 4.5f) it does not contain any vectors. This is a trade-off between those two algorithms where generally ECE-SVM provides much smaller uncertain regions, although as will be presented later this does not affect classification performance negatively (so ECE-SVM is not being overoptimistic what is considered as a certain region).

The comparison of uncertain regions of 2D datasets for those two algorithms (CE-SVM vs ECE-SVM) is presented in Figure 4.7. Comparing those images, much smaller uncertain regions, in the case of `2D-Shapes` and `2D-Blobs`, are provided by the ECE-SVM method. Although in the majority of the cases, those uncertain regions remain empty (no vectors from the whole dataset are present there). Another difference in those decision hyperplanes is their "smoothness". As ECE-SVM uses multiple cascades which are built without the knowledge of the previous level's (nodes) performance, the decision boundaries tend to be much more ragged. This is also present in those visualizations however this does not affect the final classification performance. What can be appreciated is that in the case of `2D-Dots`, both methods provide similar uncertain regions and in both cases, they cover empty regions of input space. This means that although ECE-SVM tends to provide much smaller uncertain regions, it did not lose the ability to select them. This is an important difference between those two algorithms, because CE-SVM (as shown in [41]) proved to have problems in some of the benchmark datasets, and failed to cover the test set.

Another advantage of building ensembles in the form of the cascade is their ability to adapt to new data. They also make it easier to discover new samples that come from other distributions (cover new regions of input space), with their ability to decide whether they are certain about the answer. An example of such a situation is shown in Figure 4.8. On the left-hand side (4.8a) the dataset is shown (inspired by `2D-Dots`) on which the CE-SVM method is applied. The result of training the new classifier on these data is shown in Figure 4.8b. After the CE-SVM method finishes and provides a classification model, new samples are gathered for this dataset (as often happens in real life, where data are gathered through the system lifetime). These new samples are denoted with the red circle in Figure 4.8c. Most classifiers (including SVM) would be able to classify those samples in some way, but would perform poorly in terms of the quality of such classification (probably all samples would be annotated with the positive label

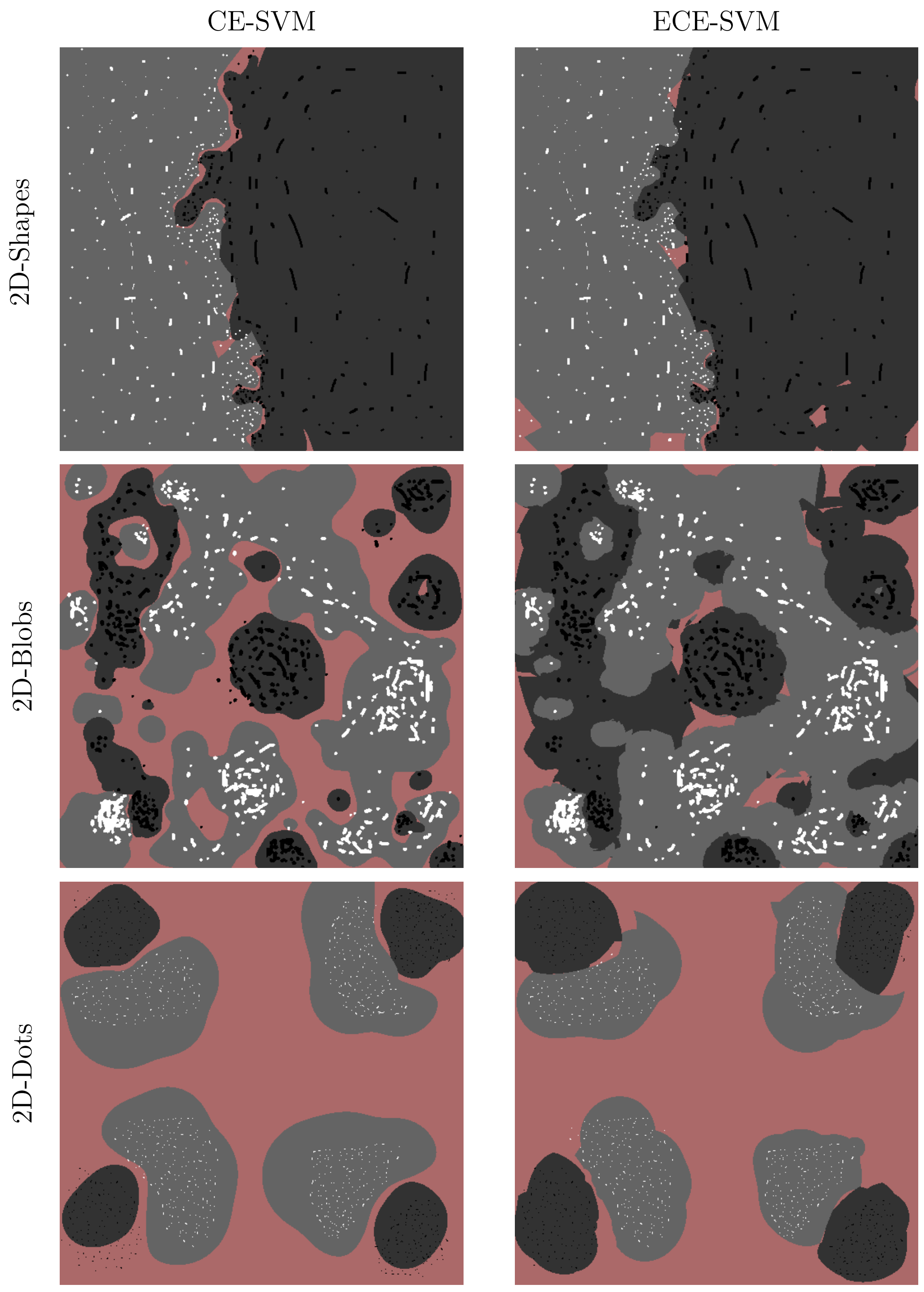


Figure 4.7: Qualitative comparisons of uncertain regions of CE-SVM and ECE-SVM ensembles.

belonging to white vectors). In the case of CE-SVM all of those samples fall into the uncertain region of the classifier. This makes them easy to detect as the model itself is able to provide a proper answer. In such a case, a procedure to update the current cascade could be made. In this particular example this results in adding a new node, that will handle this region correctly. Moreover, in Figure 4.8d, those newly added regions cover just the new data and still leave the rest of the empty space as an uncertain region. This update procedure does not require retraining the whole model. As such the update procedure could be extended in the future to cover the case of retraining single nodes that perform poorly. This could also be later extended to the ECE-SVM method although using multiple cascades would make this procedure more difficult and time-consuming. The reason is the voting scheme of ECE-SVM, where at least half of the cascades present there need to be certain, so at least half of them would need to go through an update process. On the other hand, ECE-SVM cascades are much faster to build, so it might still be viable to select just a few (best) cascades and build the rest from scratch. These update procedures might be an interesting direction for future work as they can improve practical usage of proposed methods.

Lastly, we can compare the results of those methods on a single (`2D-Blobs`) dataset presented in Figure 4.9. In the first row, there are baseline methods to compare to, the SVM(RBF) refers to a solution provided by using a grid-search algorithm to find the optimal set of hyperparameters to SVM using RBF kernel. The grid-searched solution provides great performance although it took almost 300 seconds to compute that. All other algorithms required less than 100 seconds to obtain the final result. The fastest MASVM required only 4 seconds (however it required a set of hyperparameters on input which were provided by SVM(RBF) solution). The 2D example shown here cannot be considered a difficult classification problem which is why all of the methods return high MCC values over 0.95 (except MASVM). The number of SVs remains comparable to the grid search solution (among non-ensemble solutions), where only MASVM was able to greatly reduce that number. The SE-SVM increased this number over twofold but this also resulted in an increased MCC score, compared to all other methods using a single SVM. When comparing SE-SVM to previous methods of MASVM and ALMA we can observe increased classification quality, where the decision hyperplane is more

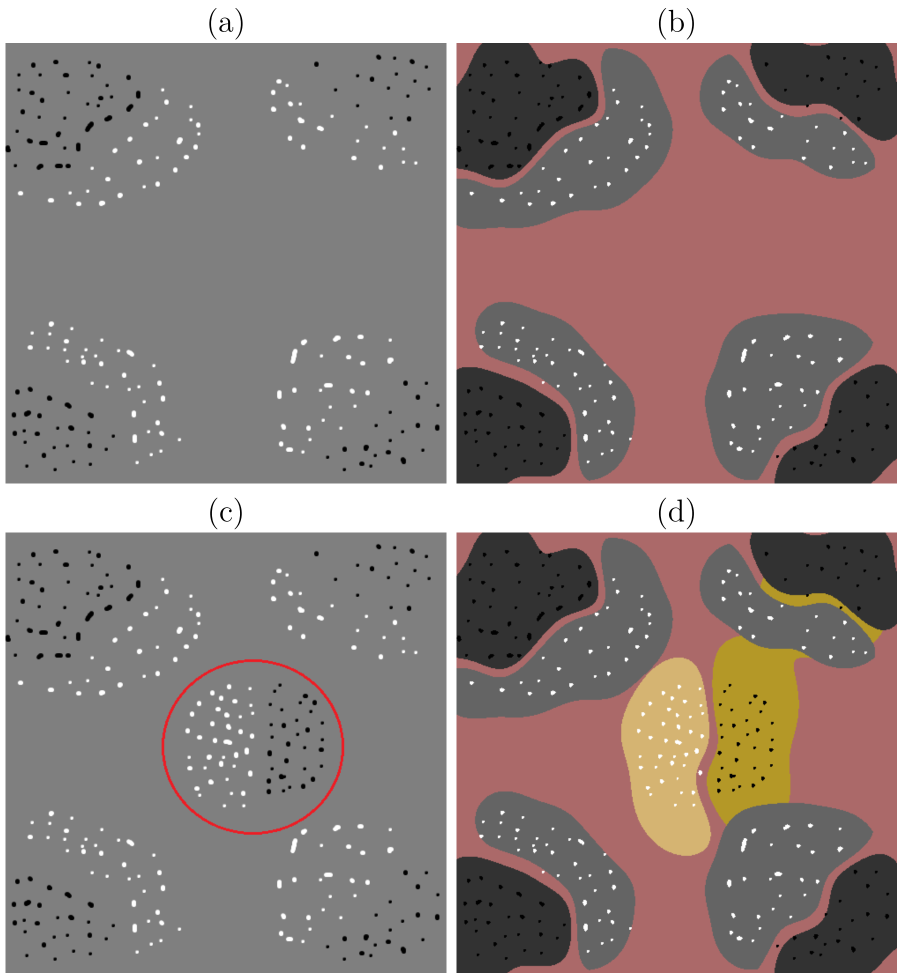


Figure 4.8: Updating the ensemble classifier: (a) shows the original dataset, (b) presents the trained CE-SVM classifier, (c) shows new data that arrived after the training, (d) shows an update to the current classifier.

detailed in the case of SE-SVM. What is interesting is that DA-SVM(CE, No-T) provided results that are on par with other methods while no SVM training was performed in this method. This proves that the training set selection can provide the $\boldsymbol{T'}$ that contains important samples which ultimately become SVs. For clarity reasons, the SVs are not visualized in the last two methods (CE-SVM and ECE-SVM). As expected, those methods using an ensemble of classifiers provided the best scores among all of the presented ones, although the number of SVs is much higher. However, this number is the sum of all vectors in the cascade or ensemble of cascades, while during the classification process, not all of them are used for a given vector. The final number of SVs used for classification depends directly on the sample and which level of cascade provides the answer.

These results are related to the first research hypothesis as qualitative analysis shows that SE-SVM classification quality is comparable with SVM(RBF) and even better than MASVM and ALMA methods. It is also shown during the analysis of behavior, that SE-SVM presents certain advantages over ALMA method, these include better initial solutions and the ability to adapt more dynamically to the problem (as both the training set and SVM hyperparameters can be tuned at the same time). The CE-SVM and ECE-SVM present interesting and unique capabilities compared to other presented algorithms such as providing an answer that they cannot decide how to classify a given sample. However, it is difficult to unambiguously show that those results provide increased classification performance in comparison to other methods (the second research hypothesis). Due to that in the next section, the quantitative analysis of 2D datasets is presented.

4.2.2 Quantitative analysis of 2D datasets for proposed algorithms

The results of experiments on 2D datasets are presented in Table 4.3 while corresponding average ranks are shown in Table 4.4. At first, it can be appreciated that all of the methods worked well and provided high-quality classification. It is the expected result, as 2D datasets are fairly simple to model and except a few datasets are fully separable (meaning that there are no examples from different classes mixed in the same area). There is a visible clear progression of results from the earliest methods (ALMA) to later ones, where presented ensembles (the latest method) provide the best scores among all of the methods. This is natural as each new method tried to improve on the drawbacks of previous ones. It is pretty interesting that DA-SVM(CE, No-T) scored the best from DA-SVM variants considering that no SVM training is performed. All of those approaches are compared with a standard grid search algorithm (SVM(RBF)). The grid search algorithm provided scores that are better than most of the evolutionary methods, except ensembles. Comparing the average results of different metrics with the average ranks of the

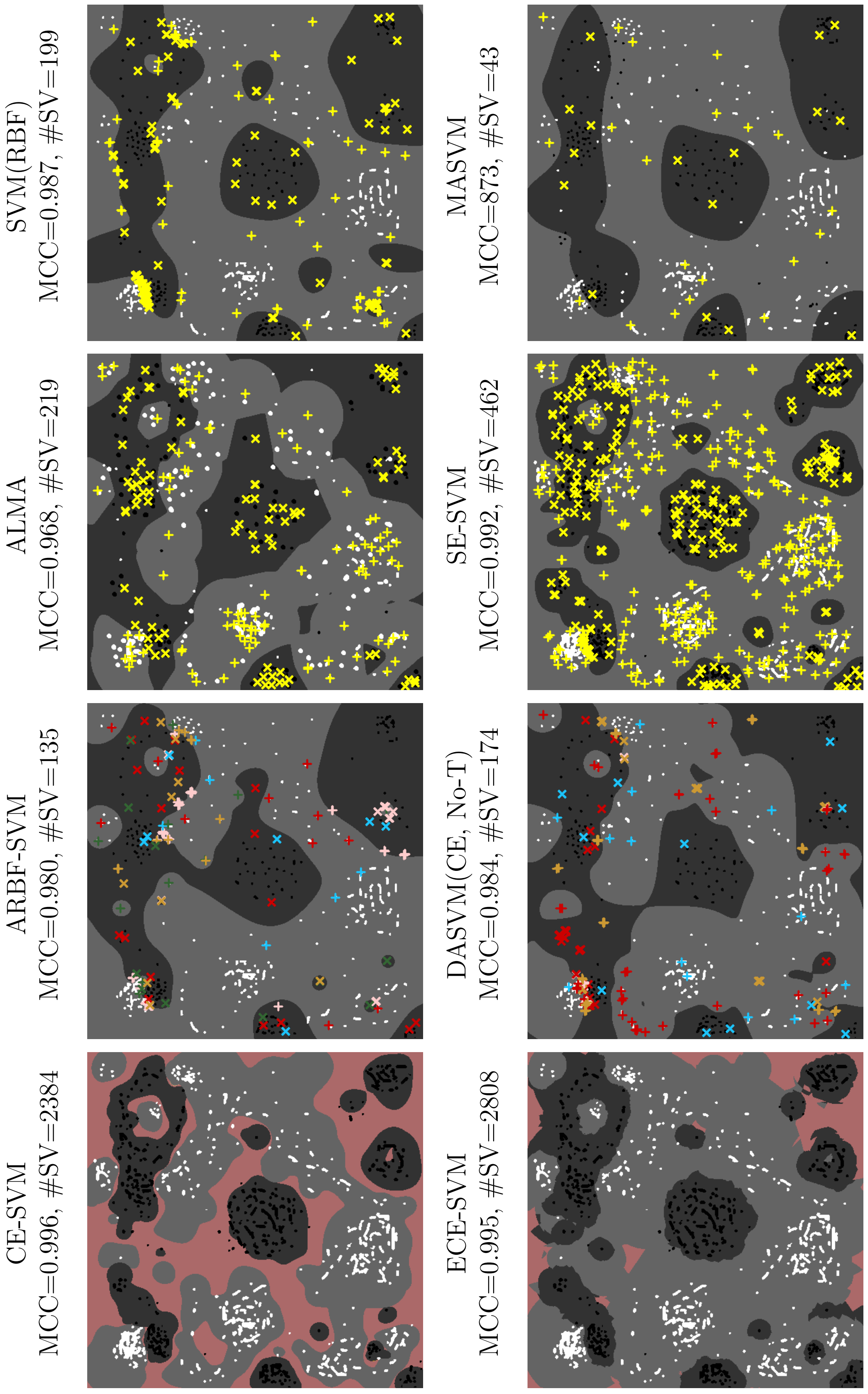


Figure 4.9: Qualititative comparions on a selected 2D dataset.

methods similar conclusion could be made. That means that there are no outliers in the performance among the datasets. The MASVM(GS) presents the results of MASVM optimization which used hyperparameters learned during the grid search algorithm (SVM(RBF)). Although it scored the lowest it was the quickest method. A more detailed analysis of the results demonstrates that in certain cases, the training set failed to grow, leading to the premature termination of the evolutionary process. This might happen as hyperparameters discovered during grid search might not be appropriate for training when K_t was low at the start. In order to better understand the performance of each method, a box plot containing the MCC score over all 2D datasets is presented in Figure 4.10. In the box plot, the differences between algorithms are easier to spot, where both proposed ensembles provided the best results outperforming all other methods. It can be observed that the methods for simultaneous optimization (SE-SVM, ARBF-SVM and DA-SVM variants), create a group, where each method gives scores similar to each other but show improvement when compared to ALMA and MASVM. On the other hand, it is also visible that comparing those methods to SVM(RBF) is not straightforward and ARBF-SVM, as well as DA-SVM(CE, MAX), are worse.

Table 4.3: Test metrics on 2D datasets. The best ones are written in bold.

	Accuracy	F1	Precision	Recall	MCC
SVM(RBF)	0.979	0.967	**0.978**	0.958	0.949
MASVM(GS)	0.932	0.886	0.917	0.878	0.837
ALMA	0.946	0.919	0.925	0.925	0.874
SE-SVM	0.973	0.959	0.959	0.961	0.937
ARBF-SVM	0.963	0.921	0.933	0.923	0.896
DA-SVM(CE,MAX)	0.960	0.926	0.935	0.926	0.897
DA-SVM(CE,No-T)	0.971	0.954	0.955	0.954	0.929
DA-SVM(CE,ONE)	0.965	0.939	0.948	0.939	0.911
CE-SVM	0.981	0.964	0.974	0.958	0.950
ECE-SVM	**0.984**	**0.971**	0.973	**0.970**	**0.958**

This result means also that the proposed DA-SVM and ARBF-SVM which introduced the adaptive kernel design might not necessarily improve the classification performance comparing them to SE-SVM which was initially hypothesized when introducing those methods. However, those results need to be confirmed on bench-

Table 4.4: Test rankings on 2D datasets. The best ones are written in bold.

	Accuracy	F1	Precision	Recall	MCC
SVM(RBF)	3.06	3.06	**1.59**	3.53	3.06
MASVM(GS)	9.24	9.24	8.53	9.18	9.24
ALMA	8.06	8.00	8.82	7.47	8.00
SE-SVM	5.12	5.24	5.12	4.71	5.18
ARBF-SVM	6.29	6.29	6.29	6.59	6.35
DA-SVM(CE, Max)	6.94	6.94	6.76	7.35	7.00
DA-SVM(CE, No-T)	5.12	5.18	4.59	5.71	5.06
DA-SVM(CE, One)	6.82	6.71	6.82	6.65	6.94
CE-SVM	2.71	2.71	2.35	2.88	2.59
ECE-SVM	**2.18**	**2.12**	2.35	**2.00**	**2.12**

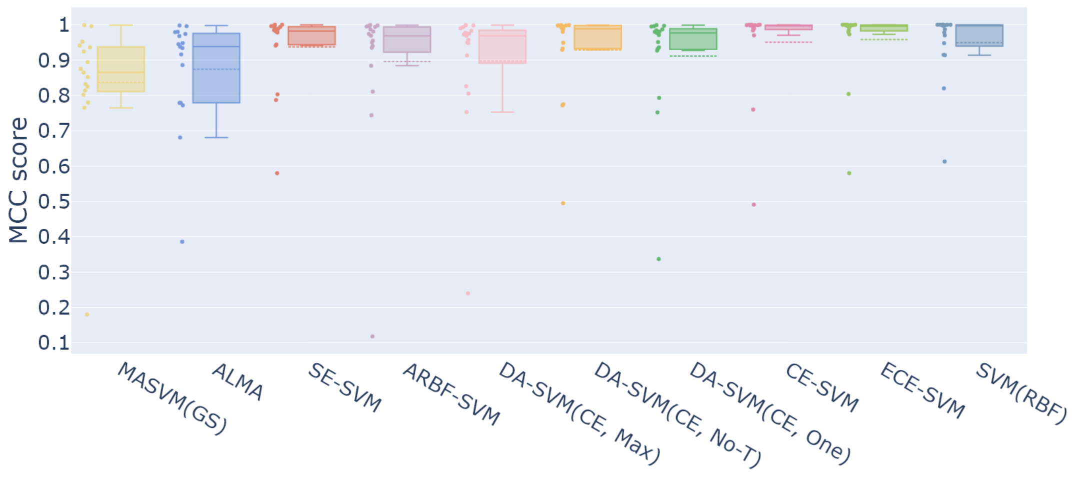


Figure 4.10: Box plot of MCC scores obtained on 2D datasets.

mark datasets to draw final conclusion. The dots on the left side indicates the results on each dataset. The ARBF-SVM, DA-SVM(CE, MAX), and MASVM(GS) show very poor performance on one dataset, which is the `2D-Mixed heavy`. That is the most imbalanced dataset among the 2D ones and hardest to classify due to the "mixing" of vectors from opposite classes in the same input space/neighborhood (see Figure 4.1). This might indicate that those methods are not well suited for such problems. Looking at the average rankings of methods (Table 4.4), they present similar results on each of the tested metric. To draw final conclusions from the data, statistical tests were performed. The Friedman test was performed and indicated statistically significant results meaning that we can reject the null hypothesis that there are no differences between the measured results of the algorithms. In order to find out detailed information the Conover post-hoc test was performed to determine which groups are significantly different from each other. The detailed results for Accuracy, F1 and MCC p-values are reported in Appendix B with Figures 1, 2, and 3. These tests are performed for those three metrics but results are almost the same in all of them (where the single exception is between DA-SVM(CE, One) and MASVM(GS) for MCC which is not statistically significant). Please note that this test only tells that there is a difference in results, but does not indicate which solution is better. This can be observed with SE-SVM on Accuracy, F1, and MCC metrics, it provides statistically different results than MASVM(GS), ALMA, CE-SVM and ECE-SVM. However, looking at results in Table 4.3 and Table 4.4, it can be observed that SE-SVM is better than MASVM(GS) and ALMA, and worse than CE-SVM and ECE-SVM. Moreover, the results of SE-SVM compared to SVM(RBF) do not provide statistical differences (the null hypothesis cannot be rejected), meaning that we did not discover differences in the results of those two algorithms (although it does not conclude that there is no difference). This directly shows that simultaneous optimization could be done without affecting the classification quality (part of the first research hypothesis). It should be noted that ensemble methods are excluded from this comparison as they use multiple base models (which use SE-SVM) to correct for mistakes among them. Nevertheless, by analyzing the results of ensembles compared to all other presented techniques, classification performance improvement can be observed. It is confirmed by the statistical tests where only non-significant result is present against SVM(RBF).

Although this lack of statistically significant improvement might also be accounted for reaching upper bound scores on the 2D datasets. On the other hand, looking at rankings (Table 4.4) it can be seen that ensembles are outperforming SVM(RBF).

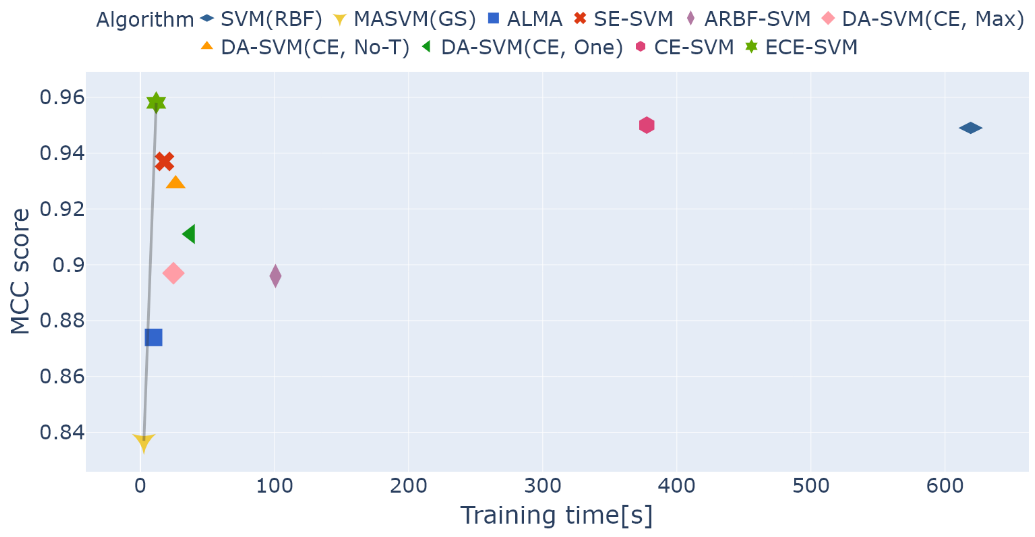


Figure 4.11: Pareto plot of MCC scores against training time. The grey line presents the Pareto front.

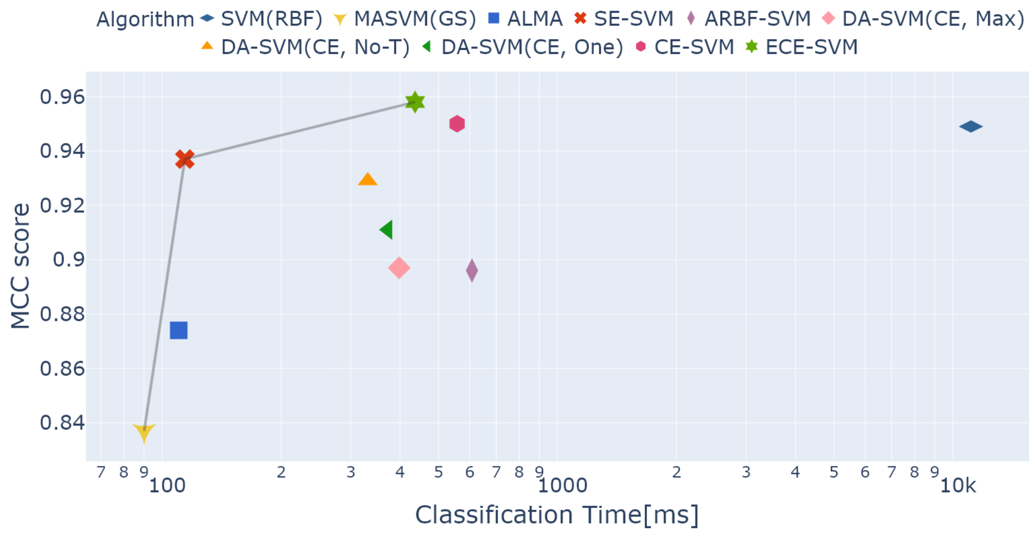


Figure 4.12: Pareto plot of MCC scores against classification time. The grey line presents the Pareto front.

However, there are two more important aspects of SVM—its training and classification time. The average values obtained by each algorithm are visualized in Figure 4.11 and Figure 4.12. These two Pareto plots show classification time and training time against the MCC metric on the test set. It is easy to spot that although SVM(GS) provides high scores with regard to classification performance in both cases it is the slowest method, often orders of magnitude slower. Please note that in this case all of the algorithms used the same implementation for training SVM classifier (LibSVM implemented in `C` version 3.19). This is important as depending on the implementation the training time could be much different (e.g. when using GPU [133]). The result in Figure 4.12 presents that the SE-SVM method is a competitive approach for classification tasks that prioritize both efficiency (i.e., low classification and training time) and effectiveness (i.e., high MCC scores). This Pareto front extends to the ECE-SVM method which provides better classification quality than SE-SVM in terms of MCC scores, but at the cost of increased classification time. This means that ECE-SVM is more suitable for applications where high classification accuracy is a priority and the cost of increased time is acceptable. Similar findings could be extracted from Figure 4.11, where the ECE-SVM is the best method, providing both the fastest training time among evolutionary methods and the highest MCC scores. Here, the SE-SVM is the second-best method in those terms, while being Pareto-optimal among techniques that consider the usage of a single SVM model. It is important to notice that most of the proposed methods provide greatly reduced training time compared to the SVM(RBF), while there is clear differences in the provided quality of classification. Considering those results it can be stated that **simultaneous optimization of the training set and the hyperparameters of the SVM improve training and classification time compared to other state-of-the-art methods proposed for this purpose without affecting the classification quality**, as shown on 2D datasets by analyzing the results of SE-SVM method. For further experiments, the MASVM(GS), ARBF-SVM, and DA-SVM(CE, MAX) methods are not analyzed, as ARBF-SVM and DA-SVM(CE, MAX) do not provide important differences compared to other DA-SVM variants, and SE-SVM, while MASVM(GS) was outperformed by all methods in quality of classification.

4.2.3 Comparison with other methods

This section will compare the performance of the best and most promising algorithms (based on 2D results analysis) to other popular classifiers and state-of-the-art algorithms for SVM optimization. At the start, a brief description will be provided for the methods used in comparison:

- The *k*-NN classifier [104] is a classification algorithm that makes predictions based on the closest data points in the feature space. Given a new data point, the k-NN classifier searches for the K closest data points in the training set and assigns the class label that is most common among those K neighbors. The choice of K is a hyperparameter that determines the number of neighbors to consider. In the presented results, three distinct values are tested K={3,5,7}.

- The random forest (RF) [104] classifier is an ensemble method that combines multiple decision trees to make predictions. Each decision tree is trained on a bootstrap sample of the training set and a random subset of the features. During prediction, the RF classifier aggregates the predictions of all the decision trees to make a final prediction. The following hyperparameters were set: Number of trees=100, criterion=Gini, Minimum samples split=2.

- The extra tree [104] classifier is another ensemble method that builds a forest of decision trees. However, unlike the RF, the ET constructs decision trees using random splits instead of the best splits. During training, the ET selects random thresholds for each feature and chooses the split that maximizes the information gain. The following hyperparameters were set: Number of trees=100, criterion=Gini, Minimum samples split=2.

- The logistic regression (LR) [104] classifier is a parametric classification algorithm that models the probability of each class using a logistic function. Given a set of input features, the LR estimates the probability of belonging to each class and assigns the class with the highest probability as the prediction. During training, the Logistic Regression classifier optimizes the parameters of the logistic function using a likelihood-based objective function.

- The gaussian naive bayes (GNB) [104] classifier is a probabilistic classification algorithm that models the class-conditional probability distribution of each feature as a Gaussian distribution. Given a set of input features, the GNB estimates the probability of belonging to each class using Bayes' theorem and assigns the class with the highest probability as the prediction. This method does not have any hyperparameters to be tuned.

- CascadeSVM [57]—The approach involves dividing the data into subsets and optimizing them separately using multiple SVMs, then combining and filtering the results through a cascade of SVMs until the global optimum is reached. However, this cascade is different then one that was presented, as SVMs are being joined throughout the process into a single SVM at the end and no expert regions are selected (the name collision is coincidental). The following hyperparameters were set: Fold size=1000, RBF kernel, SVM hyperparameters based on grid search result obtained from SVM(RBF).

- EnsembleSVM [31] is a method of building multiple SVMs trained of different subsamples of the training set. All of the base models share the same kernel and its hyperparameters. The main drawback of this method is searching for proper hyperparameters values for the size of subsamples and the number of base models that are used. The number of base models used was 100 (the same as in ET and RF methods), while for kernel the RBF was selected with $\gamma = 1/|\boldsymbol{F}| \cdot var(\boldsymbol{T})$ where $var(\cdot)$ is variance of the dataset. For size of the subsample $K_t = \{64, 128, 256\}$ were used and the best model was selected based on cross-validation score.

- Particle Swarm Optimization (PSO) [138] is a metaheuristic optimization algorithm inspired by the behavior of social organisms, such as bird flocks. In PSO, a population of particles moves through the search space seeking to optimize an objective function by updating their position and velocity based on their personal and social best solutions. In this case the PSO algorithm is used to optimize hyperparameters of SVM and select one of four kernel functions (linear, polynomial, RBF, sigmoid). The hyperparameters settings follow values presented in paper [138].

- Hyperband (HB) [138] is a randomized search strategy algorithm. The key idea behind Hyperband is to allocate more computational resources to promising configurations early in the optimization process, while quickly discarding poorly performing ones. The hyperparameters settings follow values presented in paper [138].
- Genetic algorithm [138, 80] is based on TPOT library which is an advanced tool that conducts an automated exploration of machine learning pipelines. These pipelines may include supervised classification models, preprocessors, feature selection approaches, and other estimators or transformers that comply with the Scikit-learn API. In addition, the TPOTClassifier conducts a search for the optimal hyperparameters for each pipeline component. For this study classifier was limited to SVM where C and kernel type were optimized. The hyperparameters settings follow values presented in paper [138].

Table 4.5 displays the averaged results of all presented algorithms across 31 benchmark datasets (see Section 4.1), while Table 4.6 shows their corresponding average ranks. The results reveal that SE-SVM outperforms ALMA when analyzing all metrics, which is further supported by comparing the ranking of the two methods. This demonstrates that simultaneous optimization provides better performance than the alternating approach, where different aspects are optimized independently. The statistical test (Friedman with post-hoc Conover analysis, detailed results with all p-values reported are in Appendix B) show that there is a statistically significant ($p < 0.05$) difference in MCC scores between those algorithms (it is not present for accuracy and F1 metric). When comparing SE-SVM to other methods including, literature methods designed for optimization of SVM hyperparameters like SVM(RBF), HB, PSO, or GA, and our own methods designed for the same task we can see that the average metrics stay at comparable levels. It is confirmed by analyzing the results of statistical tests, where in all of the cases for MCC metric there is no statistically significant difference between the results (excluding algorithm using ensemble techniques). This could be interpreted as classification quality is not negatively affected by using simultaneous optimization of the training set and hyperparameters of SVM which is stated by the first hypothesis. The same argument cannot be made for ALMA method which when compared to those other

Table 4.5: Results of test performed on benchmark datasets. The results are averaged scores over all of the datasets, the best ones are written in bold.

	Accuracy	F1	Precision	Recall	MCC
k-NN(3)	0.858	0.723	0.761	0.705	0.625
k-NN(5)	0.861	0.724	0.774	0.702	0.631
k-NN(7)	0.861	0.719	0.777	0.693	0.628
GNB	0.680	0.586	0.596	0.769	0.450
LR	0.854	0.694	0.787	0.661	0.610
SVM(Linear)	0.853	0.680	0.751	0.649	0.597
SVM(RBF)	**0.878**	0.734	**0.812**	0.692	0.663
HB	0.876	0.722	0.779	0.684	0.641
PSO	0.876	0.723	0.777	0.688	0.642
GA	0.843	0.689	0.740	0.655	0.606
RF	0.873	0.732	0.803	0.691	0.654
ET	0.873	0.735	0.802	0.696	0.656
EnsembleSVM	0.840	0.749	0.705	**0.851**	0.657
CascadeSVM	0.874	0.739	0.787	0.716	0.660
ALMA	0.874	0.722	0.780	0.691	0.639
SE-SVM	0.872	0.732	0.789	0.703	0.649
DA-SVM(CE, No-T)	0.863	0.711	0.777	0.677	0.623
DA-SVM(CE, One)	0.866	0.723	0.777	0.700	0.633
DA-SVM(FS, CE, No-T)	0.866	0.703	0.781	0.663	0.619
DA-SVM(FS, CE, One)	0.865	0.704	0.777	0.663	0.618
CE-SVM	0.849	0.611	0.802	0.578	0.553
ECE-SVM	**0.878**	**0.762**	0.794	0.749	**0.681**

algorithms displays worse performance. DA-SVM, on the other hand, provides average results comparable to ALMA and SE-SVM. Its variants that employ additional feature selection preprocessing perform even worse, indicating that feature selection might not be useful for the selected datasets, as it ultimately decreases the classifier's performance. Another reason could be that the feature selection algorithm (RFECV in this case) could not select better feature sets. This would need to be studied in more detail, as RFECV provides its own hyperparameters that might need more fine-tuning. Besides that, the DA-SVM (with all its variants) did not improve classification performance when compared to SE-SVM as was

Table 4.6: Ranking of methods for classification performance on benchmark datasets. The best ones are written in bold.

	Accuracy	F1	Precision	Recall	MCC
k-NN(3)	14.29	13.06	14.61	10.19	14.06
k-NN(5)	13.10	11.77	13.68	10.45	12.71
k-NN(7)	11.94	11.35	12.06	11.29	11.94
GNB	19.42	17.45	18.13	9.26	18.23
LR	8.90	9.81	9.58	11.06	9.00
SVM(Linear)	9.00	10.26	9.84	11.13	9.68
SVM(RBF)	7.87	9.32	6.06	11.45	8.23
HB	7.55	8.74	9.10	9.97	9.06
PSO	7.68	8.48	9.90	9.84	9.00
GA	8.84	9.39	10.55	10.48	10.06
RF	11.19	12.10	9.58	13.48	11.42
ET	12.00	11.84	10.13	12.45	11.32
EnsembleSVM	15.52	7.39	17.00	**4.26**	8.32
CascadeSVM	**6.84**	7.48	7.87	7.81	6.97
ALMA	13.37	14.60	12.67	14.87	14.20
SE-SVM	11.16	11.74	11.61	12.35	11.06
DA-SVM(CE, No-T)	11.48	13.52	11.90	13.90	13.23
DA-SVM(CE, One)	11.68	11.65	13.26	11.84	12.39
DA-SVM(FS, CE, No-T)	12.45	14.19	11.52	14.87	13.77
DA-SVM(FS, CE, One)	13.32	15.03	11.74	15.48	14.84
CE-SVM	15.74	16.03	9.39	16.55	16.13
ECE-SVM	8.03	**5.81**	10.13	5.55	**5.87**

hypothesized in Section 3.5. These results are confirmed with both numerical values for metrics as well as statistical tests.

Interestingly, CE-SVM performs poorly on some of the `a1a-a9a` series of datasets, resulting in significantly lower average scores compared to any of the presented methods on those datasets. This is also the reason for the much lower average scores of this method. On the rest of the benchmark datasets, the results were not alarming and on par with other tested methods. However, the reasons for such behavior are still unclear, as CE-SVM did not fail in all of the `a1a-a9a` datasets. This is strange as the general characteristics of those datasets should be similar as they represent different variants (samplings) of the same data. However, as the ECE-SVM did not show such problems, it was not studied in detail. Moreover, the ECE-SVM is designed to improve on the shortcomings of CE-SVM observed also in [41] so a more detailed analysis was not conducted. ECE-SVM, on the other hand, performs the best in F1 and MCC metrics, while considering accuracy, it is on par with SVM(RBF) which uses the grid search algorithm and full training set. As the presented benchmark sets comprise datasets with different imbalance ratio levels analyzing the results on the accuracy metric might not give the always conclusive. Comparing this method with others using statistical tests yields that ECE-SVM is the best among proposed evolutionary methods (all of the p-values for MCC and F1 scores are below 0.05). Comparing it with other methods from the literature, it can be seen that this improvement is present when analyzing F1 results (Figure 5) although, in the case of MCC metrics, the results are not statistically significant. However, this does not disapprove that there is no difference, as when comparing both the average ranking and numerical values of metric a trend of increased classification quality could be seen, where there is a clear gap between the best (ECE-SVM) and second best method. What is more, the second best method selection depends on a given metric, where for accuracy that would be SVM(RBF), for the F1 it is EnsembleSVM and for MCC score it is SVM(RBF) followed closely by CascadeSVM. SVM(Linear) and LR provide worse results as not all the data is linearly separable, so results could not be as good as other methods data can deal with non-linearly separable data (which can be regarded as more difficult to classify correctly in general). Various state-of-the-art methods for optimizing hyperparameters like PSO, HB, and GA provide good results, which are very close in relation to each other. Two methods for constructing ensembles of SVMs, EnsembleSVM and CascadeSVM, offer interesting results. EnsembleSVM faces difficulty in selecting its hyperparameters correctly as the method itself does not address the issue of hyperparameter selection. Furthermore, it adds two important hyperparameters, the subset size and the number of base models that need to be trained, thereby increasing the computational budget for checking different hyperparameter settings, which may not be practical. In contrast, CascadeSVM demonstrates to provide much better results, which are among the best in various metrics, and these results are obtained in a shorter training time compared to methods utilizing a full training set. Finally, RF and ET also offer high metrics, indicating their effectiveness in classification tasks. However, as shown in the results and statistical tests, the ECE-SVM was able to outperform them. It is important to notice that ECE-SVM also uses ET for resolving uncertain regions. Looking at the ranking, it is crucial to note that there is no single algorithm that performs best across all datasets. For instance, ECE-SVM provides the best results in F1 and MCC, but its average rank is worse than CascadeSVM and a few others when analyzing accuracy. However, the differences are noticeable when comparing the training and classification times among the SVM methods as presented in Figure 4.13 and Figure 4.14. It can be appreciated that in both cases the Pareto-optimal solutions are provided by methods proposed in this work. For classification time, the Pareto-optimal solution is created by ALMA, SE-SVM, and ECE-SVM while for training time those are ALMA and ECE-SVM. In the case of training time, the time performance of SE-SVM might be impacted by the feature selection method which was additionally tested here. These insights highlight the importance of selecting a method that is tailored to the specific characteristics and requirements of the given dataset.

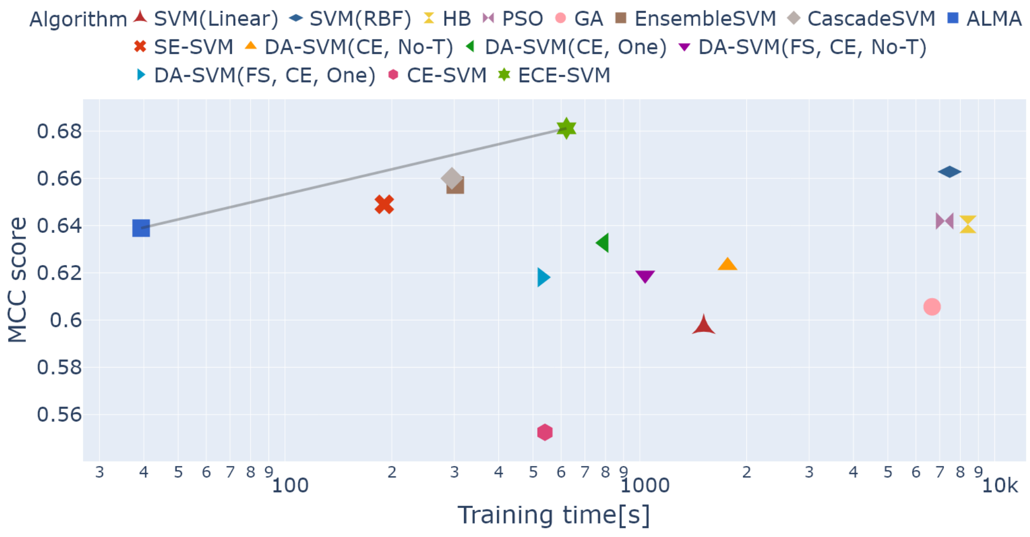


Figure 4.13: Pareto plot of MCC scores against training time for SVM methods.

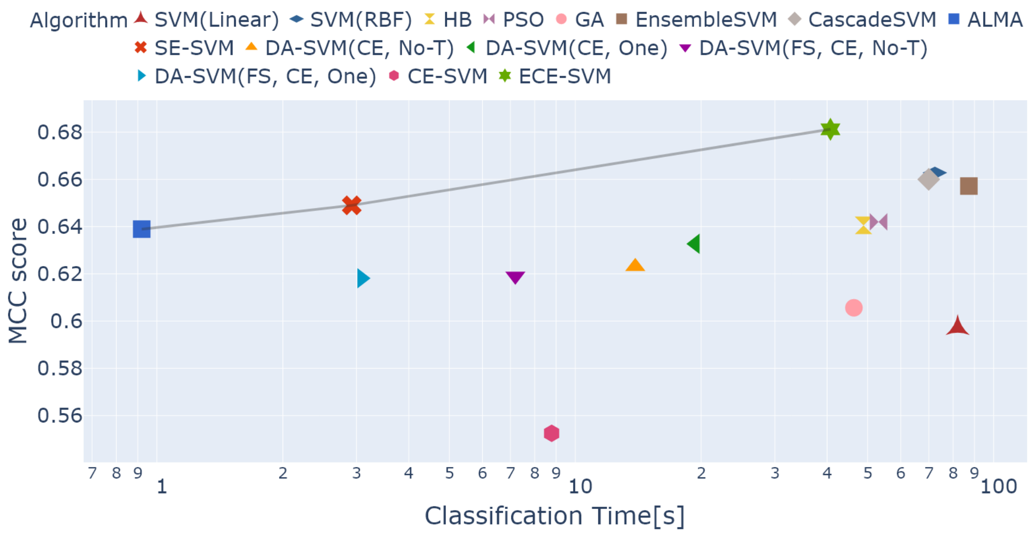


Figure 4.14: Pareto plot of MCC scores against classification time for SVM methods.

4.3 Summary

In summary, based on the presented results it has been confirmed that SE-SVM which performs simultaneous optimization of the training set and the hyperparameters of the SVM could provide improved training and classification times without affecting the classification performance. This research hypothesis is supported by qualitative analysis and comparison of SE-SVM with ALMA and DA-SVM methods. Furthermore, in the quantitative analysis performed on 2D datasets the SE-SVM proved to be significantly better than ALMA and MASVM methods, at the same time its performance is comparable with the models obtained by running grid search optimization over a full training set. Analyzing classification times on both benchmark and 2D datasets SE-SVM displays a Pareto-optimal solution (compared using MCC metric). For training times in both cases, it needs more time compared to ALMA, although increased classification performance can justify prolonged times, where SE-SVM lies close to the Pareto front. The quantitative analysis of 31 benchmark datasets shows that SE-SVM does not trade-off classification quality when compared to other methods using a single SVM model and is outperformed mainly by ensemble methods, both from literature and proposed ones. Regarding the second hypothesis that building ensembles of SVM using evolutionary algorithms provide improved classification performance, it is shown in quantitative analysis in both 2D and benchmark datasets. On qualitative analysis, there are highlights of other interesting advantages such as the possibility to update existing classifier systems upon arrival of new data, where it might not be needed to train a new model from the ground up, and providing post-hoc analysis of the characteristics of the datasets. The latter could be done by analyzing data in certain and uncertain regions of such a model. The results presented on 2D benchmark sets show that both CE-SVM and ECE-SVM provide increased classification performance when compared to other evolutionary methods which are supported by statistical tests. The lack of statistical significance between those methods and SVM(RBF) is due to hitting the upper bound of classification performance, where all metrics are close to perfect classification scores on the test sets so further improvement is not possible. However, there are visible trends in both numerical results and average rankings when comparing those methods, where ECE-SVM presents the best results in all of the metrics. A similar situation (with elevated numerical and rank scores) is presented in 31 benchmark datasets where ECE-SVM provided the best scores among all presented methods in F1 and MCC (for average results and rankings) as well as the best average accuracy (on par with SVM(RBF)). The statistical tests confirmed that in multiple cases this difference is significant. What is more, ECE-SVM provided the Pareto-optimal solution in all of the tests regarding classification and training times coupled to MCC metric. All of these results positively verify the second research hypothesis.

Chapter 5

Conclusions

The focus of this dissertation was on the optimization of the training set, and hyperparameters for SVMs by using evolutionary computation and building different forms of classification ensembles. In the beginning, the SVM and the theory behind it were introduced and the literature was reviewed. Based on that, it could be spotted that optimizing multiple aspects of SVM simultaneously could be researched further to improve existing methods. The optimization of these parameters is a challenging task, and different approaches have been proposed to address this problem, while there was no method available to tackle all of them. This research proposes novel solutions that combine the optimization of these aspects ($\mathcal{M}$, $\boldsymbol{T}$) using alternating and simultaneous optimization approaches. Those algorithms were introduced in detail in Chapter 3. All of them are based on evolutionary computations as they demonstrated to be efficient in finding reduced training sets as seen in MASVM [96]. This method serves as a baseline for finding and optimizing $\boldsymbol{T'}$, while the new algorithms for optimizing hyperparameters and feature sets are proposed. These methods are then combined into alternating optimization and finally, simultaneous optimization is proposed. One of the advantages of developing such approaches is that each of those components (optimizing only one of $\mathcal{M}$, $\boldsymbol{T}$, $\boldsymbol{F}$) could be easily replaced as demonstrated with ALGA and ALMA algorithms. The latter uses an improved MASVM method (over the original GASVM), where the whole scheme of alternating optimization is unchanged. With some additional work, the same approach could be used in SE-SVM algorithm which brings interesting direction to future research, where feature selection could be studied in more detail and other components of the algorithm could be exchanged. Then the algorithms are analyzed experimentally in Section 4.2, where the results of these approaches are discussed and visualized using artificial datasets. The methods are also compared to other popular classifiers and state-of-the-art SVM optimization techniques using benchmark datasets.

The first thesis stating that **simultaneous optimization of the training set and the SVM hyperparameters improve training and classification time compared to other state-of-the-art methods proposed for this purpose without affecting the classification quality** is demonstrated by the SE-SVM method compared to ALMA and MASVM methods. Analyzing both the qualitative and quantitative results clearly show the SE-SVM approach can better utilize the mutual dependence of optimizing $\mathcal{M}$, $\boldsymbol{T}$ and improve classifier performance compared to the alternating optimization techniques. What is more, the training and classification time were also analyzed. First of all those times are greatly reduced compared to the regular grid search algorithm in both cases of ALMA and SE-SVM. Although ALMA algorithm had faster training and classification time it has worse classification performance when compared to SE-SVM. This means that SE-SVM is a Pareto-optimal method that positively verifies this hypothesis. Moreover, the DASVM provided promising results on 2D datasets but ultimately this method did not perform so well in benchmark settings, where its quality is between ALMA and SE-SVM methods but training and classification times are longer.

The second thesis stating that **SVM ensembles created using evolutionary algorithms provide improved classification performance compared to other well-established methods, including existing algorithms for building SVM ensembles** is addressed with CE-SVM and ECE-SVM algorithms. These algorithms divide the dataset into different regions, and by building these regions into the cascade structure of classifiers, the SVMs performance and applicability are improved. One of the improvements is the introduction the concept of certain regions. The idea behind this approach is to identify regions within the dataset where the SVM can make accurate predictions with high confidence. Studies have shown that such expert regions can indeed be identified and that they can be classified with higher accuracy than the entire dataset. While this approach may require a longer classification time, the results have been shown to be superior to other methods. It is important to note that although the training time for SVMs with expert regions may be increased compared to other methods such as ALMA and SE-SVM, the increased training time is a natural consequence of building multiple models to identify and focus on expert regions. What is more, those results are also Pareto-optimal meaning that the longer training and classification time is associated with improved classification performance. Overall, incorporating the concept of expert regions into SVMs is a promising approach for improving their performance, particularly in scenarios where classification performance and confidence in predictions are crucial. While the approach may require more computational resources, the potential gains in classification quality and confidence make it a valuable technique for further usage.

5.1 Future work

The presented results and methods could be used as a great starting point for further research. There are numerous things that were introduced and tested while some of those seem (in the author's opinion) to be too briefly touched on.

The optimization of hyperparameters is a critical aspect of SVMs, as the performance of the model is highly dependent on the choice of hyperparameters. The hyperparameter optimization method proposed in this research is fairly simple, and coupling proposed algorithms with more advanced techniques could potentially lead to further improvements. For instance, the use of Bayesian optimization or gradient-based optimization techniques could lead to more efficient and accurate hyperparameter tuning. Additionally, incorporating prior knowledge or domain expertise into the optimization process could also be explored as a way to guide the search towards more promising hyperparameter configurations. Yet another interesting direction is finding promising hyperparametrs by the greatly reducing size of the training set as shown with initial experimentation in Section 3.5 where grid search was used.

This research has been focused on binary classification problems, and extending it into multiclassification problems using specialized SVMs is another interesting direction. As shown in earlier work [94], using one-vs-one strategy to tackle those is not efficient and state-of-the-art methods are better. Multi-class classification problems are commonly encountered in many applications, and developing efficient and accurate methods for solving these problems is an important research area. The extension of the proposed algorithms to handle Multi-class classification problems could lead to new insights and improvements in performance. This should not be specifically hard to follow as there is no inherent limitation to the number of classes handled by the evolutionary method proposed.

Coupling ensemble methods with feature selection is another interesting direction for future research. Ensemble methods are known to improve the accuracy and robustness of machine learning models by combining multiple models' predictions. Feature selection, on the other hand, is a technique that selects the most relevant features from the dataset to improve the model's performance and reduce overfitting. Combining ensemble methods with feature selection could lead to further improvements in performance by selecting the most relevant features and increasing the diversity of included models. What is more, this could make CE-SVM and ECE-SVM easier to use for non-expert users.

Lastly, the DA-SVM algorithm with the No-T variant presented in this research is an interesting approach that could be extended to other algorithms presented here. The selection of a small yet representative validation set is crucial for this algorithm to be effective, as it is used to estimate the fitness of the classifier and could be considered as a replacement for the training process. Further research could explore the use of the No-T variant in other algorithms and investigate the impact of the validation set size on performance.

Bibliography

[1] Abdel-Basset, M., Ding, W., and El-Shahat, D. A hybrid Harris Hawks optimization algorithm with simulated annealing for feature selection. *Artificial Intelligence Review 54* (2021), 593–637.

[2] Abe, S., and Inoue, T. Fast training of support vector machines by extracting boundary data. In *Proceedings of International Conference on Artificial Neural Networks* (2001), Springer, pp. 308–313.

[3] Abusitta, A., Li, M. Q., and Fung, B. C. Malware classification and composition analysis: A survey of recent developments. *Journal of Information Security and Applications 59* (2021), 102828.

[4] Aiolli, F., and Donini, M. EasyMKL: a scalable multiple kernel learning algorithm. *Neurocomputing 169* (2015), 215–224.

[5] Aladeemy, M., Tutun, S., and Khasawneh, M. T. A new hybrid approach for feature selection and support vector machine model selection based on self-adaptive cohort intelligence. *Expert Systems with Applications 88* (2017), 118–131.

[6] Albashish, D., Hammouri, A. I., Braik, M., Atwan, J., and Sahran, S. Binary biogeography-based optimization based SVM-RFE for feature selection. *Applied Soft Computing 101* (2021), 107026.

[7] Angiulli, F., and Astorino, A. Scaling up support vector machines using nearest neighbor condensation. *IEEE Transactions on Neural Networks 21*, 2 (2010), 351–357.

[8] Aslan, M. F., Sabanci, K., Durdu, A., and Unlersen, M. F. Covid-19 diagnosis using state-of-the-art CNN architecture features and Bayesian Optimization. *Computers in Biology and Medicine 142* (2022), 105244.

[9] Ayat, N., Cheriet, M., and Suen, C. Automatic model selection for the optimization of SVM kernels. *Pattern Recognition 38*, 10 (2005), 1733–1745.

[10] Ayush, K., and Sinha, A. Improving classification performance of support vector machines via guided custom kernel search. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion* (New York, NY, USA, 2019), pp. 159—-160.

[11] Balcázar, J., Dai, Y., and Watanabe, O. A random sampling technique for training support vector machines. In *Proceedings of Algorithmic Learning Theory* (Berlin, Heidelberg, 2001), Springer, pp. 119–134.

[12] Baliarsingh, S. K., Ding, W., Vipsita, S., and Bakshi, S. A memetic algorithm using emperor penguin and social engineering optimization for medical data classification. *Applied Soft Computing 85* (2019), 105773.

[13] Benkessirat, A., and Benblidia, N. Fundamentals of feature selection: An overview and comparison. In *Proceedings of International Conference on Computer Systems and Applications* (2019), pp. 1–6.

[14] Bergstra, J., Bardenet, R., Bengio, Y., and Kégl, B. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems 24* (2011).

[15] Bergstra, J., and Bengio, Y. Random search for hyper-parameter optimization. *Journal of Machine Learning Research 13*, 10 (2012), 281–305.

[16] Bergstra, J., Yamins, D., and Cox, D. D. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *Proceedings of the 30th International Conference on Machine Learning* (2013), ICML'13, pp. 115—-123.

[17] Bischl, B., Binder, M., Lang, M., Pielok, T., Richter, J., Coors, S., Thomas, J., Ullmann, T., Becker, M., Boulesteix, A.-L., et al. Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* (2021), e1484.

[18] Bommert, A., Sun, X., Bischl, B., Rahnenführer, J., and Lang, M. Benchmark for filter methods for feature selection in high-dimensional classification data. *Computational Statistics & Data Analysis 143* (2020), 106839.

[19] Boser, B. E., Guyon, I. M., and Vapnik, V. N. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory* (New York, NY, USA, 1992), pp. 144—-152.

[20] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems 33* (2020), 1877–1901.

[21] Carcillo, F., Le Borgne, Y.-A., Caelen, O., Kessaci, Y., Oblé, F., and Bontempi, G. Combining unsupervised and supervised learning in credit card fraud detection. *Information sciences 557* (2021), 317–331.

[22] Cervantes, J., Lamont, F. G., López-Chau, A., Mazahua, L. R., and Ruíz, J. S. Data selection based on decision tree for SVM classification on large data sets. *Applied Soft Computing 37* (2015), 787–798.

[23] Chandrashekar, G., and Sahin, F. A survey on feature selection methods. *Computers & Electrical Engineering 40*, 1 (2014), 16–28.

[24] Chang, C.-C., and Lin, C.-J. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology 2*, 3 (2011), 1–27.

[25] Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research 16* (2002), 321–357.

[26] Chen, C., Li, X., Belkacem, A. N., Qiao, Z., Dong, E., Tan, W., and Shin, D. The mixed kernel function SVM-based point cloud classification. *International Journal of Precision Engineering and Manufacturing 20*, 5 (2019), 737–747.

[27] Chen, L., Li, S., Bai, Q., Yang, J., Jiang, S., and Miao, Y. Review of image classification algorithms based on convolutional neural networks. *Remote Sensing 13*, 22 (2021), 4712.

[28] Cheng, F., Chen, J., Qiu, J., and Zhang, L. A subregion division based multi-objective evolutionary algorithm for SVM training set selection. *Neurocomputing 394* (2020), 70–83.

[29] Chicco, D., and Jurman, G. The advantages of the matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. *BMC Genomics 21*, 1 (Jan 2020), 6.

[30] Chicco, D., Tötsch, N., and Jurman, G. The Matthews correlation coefficient (MCC) is more reliable than balanced accuracy, bookmaker informedness, and markedness in two-class confusion matrix evaluation. *BioData Mining 14*, 1 (Feb 2021), 13.

[31] Claesen, M., De Smet, F., Suykens, J. A. K., and De Moor, B. Ensemblesvm: A library for ensemble learning using support vector machines. *Journal of Machine Learning Research 15*, 1 (jan 2014), 141–145.

[32] Cortes, C., and Vapnik, V. Support vector networks. *Machine Learning 20* (1995), 273–297.

[33] Czarnowski, I. Cluster-based instance selection for machine classification. *Knowledge and Information Systems 30*, 1 (2012), 113–133.

[34] Demšar, J. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research 7* (2006), 1–30.

[35] Dhiman, H. S., Deb, D., Muyeen, S., and Kamwa, I. Wind turbine gearbox anomaly detection based on adaptive threshold and twin support vector machines. *IEEE Transactions on Energy Conversion 36*, 4 (2021), 3462–3469.

[36] Dixon, M. F., Halperin, I., and Bilokon, P. *Machine learning in Finance*, vol. 1170. Springer, 2020.

[37] Dudzik, W., Kawulok, M., and Nalepa, J. Evolutionarily-tuned support vector machines. In *Proceedings of Genetic and Evolutionary Computation Conference Companion* (New York, NY, USA, 2019), pp. 165–166.

[38] Dudzik, W., Kawulok, M., and Nalepa, J. Optimizing training data and hyperparameters of support vector machines using a memetic algorithm. In *Proceedings of International Conference on Man-Machine Interactions* (2019), pp. 229–238.

[39] Dudzik, W., Nalepa, J., and Kawulok, M. Automated optimization of non-linear support vector machines for binary classification. In *Proceedings of Advances in Intelligent Networking and Collaborative Systems* (Cham, 2019), pp. 504–513.

[40] Dudzik, W., Nalepa, J., and Kawulok, M. Evolving data-adaptive support vector machines for binary classification. *Knowledge-Based Systems 227* (2021), 107221.

[41] Dudzik, W., Nalepa, J., and Kawulok, M. Cascades of evolutionary support vector machines. In *Proceedings of Genetic and Evolutionary Computation Conference Companion* (2022), pp. 240–243.

[42] Falkner, S., Klein, A., and Hutter, F. BOHB: Robust and efficient hyperparameter optimization at scale. In *Proceedings of the 35th International Conference on Machine Learning* (Jul 2018), vol. 80, pp. 1437–1446.

[43] Faris, H., Hassonah, M. A., Al-Zoubi, A. M., Mirjalili, S., and Aljarah, I. A multi-verse optimizer approach for feature selection and

optimizing SVM parameters based on a robust system architecture. *Neural Computing and Applications 30*, 8 (Oct 2018), 2355–2369.

[44] FAWCETT, T. An introduction to ROC analysis. *Pattern Recognition Letters 27*, 8 (2006), 861–874.

[45] FLOREA, A. C., AND ANDONIE, R. A dynamic early stopping criterion for random search in SVM hyperparameter optimization. In *Proceedings of International Conference on Artificial Intelligence Applications and Innovations* (2018), Springer, pp. 168–180.

[46] FRIEDMAN, J. H. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics 29*, 5 (2001), 1189 – 1232.

[47] GAO, L., QIAN, W., LI, X., AND WANG, J. Application of memetic algorithm in assembly sequence planning. *The International Journal of Advanced Manufacturing Technology 49*, 9 (Aug 2010), 1175–1184.

[48] GARCIA, S., DERRAC, J., CANO, J., AND HERRERA, F. Prototype selection for nearest neighbor classification: Taxonomy and empirical study. *IEEE transactions on pattern analysis and machine intelligence 34*, 3 (2012), 417–435.

[49] GARCÍA, V., MOLLINEDA, R. A., AND SÁNCHEZ, J. S. Index of balanced accuracy: A performance measure for skewed class distributions. In *Pattern Recognition and Image Analysis* (Berlin, Heidelberg, 2009), Springer, pp. 441–448.

[50] GARCÍA-PEDRAJAS, N., DE HARO-GARCÍA, A., AND PÉREZ-RODRÍGUEZ, J. A scalable memetic algorithm for simultaneous instance and feature selection. *Evolutionary Computation 22*, 1 (2014), 1–45.

[51] GEURTS, P., ERNST, D., AND WEHENKEL, L. Extremely randomized trees. *Machine Learning 63*, 1 (Apr 2006), 3–42.

[52] GHAMISI, P., AND BENEDIKTSSON, J. A. Feature selection based on hybridization of genetic algorithm and particle swarm optimization. *IEEE Geoscience and Remote Sensing Letters 12*, 2 (Feb 2015), 309–313.

[53] Ghamisi, P., Couceiro, M. S., and Benediktsson, J. A. A novel feature selection approach based on FODPSO and SVM. *IEEE Transactions on Geoscience and Remote Sensing 53*, 5 (May 2015), 2935–2947.

[54] Goel, A., and Srivastava, S. K. Role of kernel parameters in performance evaluation of SVM. In *Proceedings of Computational Intelligence & Communication Technology* (2016), IEEE, pp. 166–169.

[55] Gomez-Uribe, C. A., and Hunt, N. The netflix recommender system: Algorithms, business value, and innovation. *ACM Transactions on Management Information Systems 6*, 4 (2015), 1–19.

[56] Gopi, A. P., Jyothi, R. N. S., Narayana, V. L., and Sandeep, K. S. Classification of tweets data based on polarity using improved RBF kernel of SVM. *International Journal of Information Technology* (Jan 2020).

[57] Graf, H., Cosatto, E., Bottou, L., Dourdanovic, I., and Vapnik, V. Parallel support vector machines: The Cascade SVM. In *Advances in Neural Information Processing Systems* (2004), vol. 17.

[58] Guo, L., and Boukir, S. Fast data selection for SVM training using ensemble margin. *Pattern Recognition Letters 51* (2015), 112–119.

[59] Guo, L., Boukir, S., and Chehata, N. Support vectors selection for supervised learning using an ensemble approach. In *Proceedings of 20th International Conference on Pattern Recognition* (2010), pp. 37–40.

[60] Guyon, I., Weston, J., Barnhill, S., and Vapnik, V. Gene selection for cancer classification using support vector machines. *Machine Learning 46*, 1 (Jan 2002), 389–422.

[61] Herrera, F., Lozano, M., and Verdegay, J. L. Tackling real-coded genetic algorithms: Operators and tools for behavioural analysis. *Artificial Intelligence Review 12*, 4 (Aug 1998), 265–319.

[62] Hofmann, T., Schölkopf, B., and Smola, A. J. Kernel methods in machine learning. *The Annals of Statistics 36*, 3 (2008), 1171 – 1220.

[63] Holland, J. H. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence.* MIT Press, 1992.

[64] Houssein, E. H., Hosney, M. E., Oliva, D., Mohamed, W. M., and Hassaballah, M. A novel hybrid harris hawks optimization and support vector machines for drug design and discovery. *Computers & Chemical Engineering 133* (2020), 106656.

[65] Hsu, C.-W., Chang, C.-C., and Lin, C.-J. A practical guide to support vector classification. Tech. rep., Department of Computer Science, National Taiwan University, 2003.

[66] Huang, C.-L., and Dun, J.-F. A distributed PSO–SVM hybrid system with feature selection and parameter optimization. *Applied Soft Computing 8*, 4 (2008), 1381–1391.

[67] Huang, S., Cai, N., Pacheco, P. P., Narrandes, S., Wang, Y., and Xu, W. Applications of support vector machine (SVM) learning in cancer genomics. *Cancer Genomics Proteomics 15*, 1 (Jan. 2018), 41–51.

[68] Huanrui, H. New mixed kernel functions of SVM used in pattern recognition. *Cybernetics and Information Technologies 16*, 5 (2016), 5–14.

[69] Hussain, S. F. A novel robust kernel for classifying high-dimensional data using support vector machines. *Expert Systems with Applications 131* (2019), 116–131.

[70] Hutter, F., Hoos, H. H., and Leyton-Brown, K. Sequential model-based optimization for general algorithm configuration. In *Proceedings of the 5th International Conference on Learning and Intelligent Optimization* (Berlin, Heidelberg, 2011), p. 507–523.

[71] Ibrahim, H. T., Mazher, W. J., Ucan, O. N., and Bayat, O. A grasshopper optimizer approach for feature selection and optimizing SVM parameters utilizing real biomedical data sets. *Neural Computing and Applications 31* (2019), 5965–5974.

[72] Jamieson, K., and Talwalkar, A. Non-stochastic best arm identification and hyperparameter optimization. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics* (May 2016), vol. 51, PMLR, pp. 240–248.

[73] Jiang, H., Ching, W.-K., Yiu, K. F. C., and Qiu, Y. Stationary mahalanobis kernel SVM for credit risk evaluation. *Applied Soft Computing 71* (2018), 407–417.

[74] Katoch, S., Chauhan, S. S., and Kumar, V. A review on genetic algorithm: Past, present, and future. *Multimedia Tools and Applications 80*, 5 (Feb 2021), 8091–8126.

[75] Kawulok, M., and Nalepa, J. Support vector machines training data selection using a genetic algorithm. In *Structural, Syntactic, and Statistical Pattern Recognition* (Berlin, Heidelberg, 2012), pp. 557–565.

[76] Kawulok, M., Nalepa, J., and Dudzik, W. An alternating genetic algorithm for selecting SVM model and training set. In *Proceedings of Mexican Conference on Pattern Recognition* (2017), Springer, pp. 94–104.

[77] Klein, A., Falkner, S., Bartels, S., Hennig, P., and Hutter, F. Fast Bayesian Optimization of machine learning hyperparameters on large datasets. In *Artificial intelligence and statistics* (2017), PMLR, pp. 528–536.

[78] Kuo, B.-C., Ho, H.-H., Li, C.-H., Hung, C.-C., and Taur, J.-S. A kernel-based feature selection method for SVM with RBF kernel for hyperspectral image classification. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing 7*, 1 (2013), 317–326.

[79] Kurani, A., Doshi, P., Vakharia, A., and Shah, M. A comprehensive comparative study of artificial neural network (ANN) and support vector machines (SVM) on stock forecasting. *Annals of Data Science 10*, 1 (Feb 2023), 183–208.

[80] Le, T. T., Fu, W., and Moore, J. H. Scaling tree-based automated machine learning to biomedical big data with a feature set selector. *Bioinformatics 36*, 1 (2020), 250–256.

[81] Lessmann, S., Stahlbock, R., and Crone, S. F. Genetic algorithms for support vector machine model selection. In *Proceedings of IEEE International Joint Conference on Neural Network* (2006), IEEE, pp. 3063–3069.

[82] Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research 18*, 1 (2017), 6765–6816.

[83] Lin, C.-C., Kang, J.-R., Liang, Y.-L., and Kuo, C.-C. Simultaneous feature and instance selection in big noisy data using memetic variable neighborhood search. *Applied Soft Computing 112* (2021), 107855.

[84] Louppe, G., and Geurts, P. Ensembles on random patches. In *Machine Learning and Knowledge Discovery in Databases* (Berlin, Heidelberg, 2012), Springer, pp. 346–361.

[85] Lu, C., Zhang, B., Gao, L., Yi, J., and Mou, J. A knowledge-based multiobjective memetic algorithm for green job shop scheduling with variable machining speeds. *IEEE Systems Journal 16*, 1 (2021), 844–855.

[86] Maleki, N., Zeinali, Y., and Niaki, S. T. A. A k-NN method for lung cancer prognosis with the use of a genetic algorithm for feature selection. *Expert Systems with Applications 164* (2021), 113981.

[87] Meinshausen, N., and Bühlmann, P. Stability selection. *Journal of the Royal Statistical Society: Series B (Statistical Methodology) 72*, 4 (2010), 417–473.

[88] Menezes, M. V., Torres, L. C., and Braga, A. P. Width optimization of RBF kernels for binary classification of support vector machines: A density estimation-based approach. *Pattern Recognition Letters 128* (2019), 1–7.

[89] Michalewicz, Z. *Genetic algorithms + data structures = evolution programs (3nd, extended ed.).* Springer-Verlag, New York, NY, USA, 1996.

[90] Mitchell, M. Genetic algorithms: An overview. In *Complex.* (1995), vol. 1, Citeseer, pp. 31–39.

[91] Molina, J. C., Salmeron, J. L., and Eguia, I. An ACS-based memetic algorithm for the heterogeneous vehicle routing problem with time windows. *Expert Systems with Applications 157* (2020), 113379.

[92] Moscato, P. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Tech. Rep. C3P Report 826, California Institute of Technology, 1989.

[93] Moscato, P., and Mathieson, L. *Memetic Algorithms for Business Analytics and Data Science: A Brief Survey.* Springer International Publishing, Cham, 2019, pp. 545–608.

[94] Nalepa, J., Dudzik, W., and Kawulok, M. Memetic evolution of training sets with adaptive radial basis kernels for support vector machines. In *Proceedings of International Conference on Pattern Recognition* (2020), pp. 1–8.

[95] Nalepa, J., and Kawulok, M. Adaptive genetic algorithm to select training data for support vector machines. In *Applications of Evolutionary Computation* (Berlin, Heidelberg, 2014), Springer, pp. 514–525.

[96] Nalepa, J., and Kawulok, M. A memetic algorithm to select training data for support vector machines. In *Proceedings of Genetic and Evolutionary Computation Conference* (New York, NY, USA, 2014), pp. 573–580.

[97] Nalepa, J., and Kawulok, M. Adaptive memetic algorithm enhanced with data geometry analysis to select training data for SVMs. *Neurocomputing 185* (2016), 113–132.

[98] Nalepa, J., and Kawulok, M. Selecting training sets for support vector machines: A review. *Artificial Intelligence Review 52*, 2 (2019), 857–900.

[99] Nalepa, J., Siminski, K., and Kawulok, M. Towards parameter-less support vector machines. In *Proceedings of Asian Conference on Pattern Recognition* (2015), pp. 211–215.

[100] NANGLIA, S., AHMAD, M., ALI KHAN, F., AND JHANJHI, N. An enhanced predictive heterogeneous ensemble model for breast cancer prediction. *Biomedical Signal Processing and Control 72* (2022), 103279.

[101] NEUMANN, J., SCHNÖRR, C., AND STEIDL, G. Combined SVM-based feature selection and classification. *Machine Learning 61*, 1 (Nov 2005), 129–150.

[102] OGUTU, J. O., SCHULZ-STREECK, T., AND PIEPHO, H.-P. Genomic selection using regularized linear regression models: Ridge regression, LASSO, elastic net and their extensions. *BMC Proceedings 6*, 2 (May 2012), S10.

[103] PANCH, T., SZOLOVITS, P., AND ATUN, R. Artificial intelligence, machine learning and health systems. *Journal of Global Health 8*, 2 (Dec. 2018), 020303.

[104] PEDREGOSA, F., VAROQUAUX, G., GRAMFORT, A., MICHEL, V., THIRION, B., GRISEL, O., BLONDEL, M., PRETTENHOFER, P., WEISS, R., DUBOURG, V., VANDERPLAS, J., PASSOS, A., COURNAPEAU, D., BRUCHER, M., PERROT, M., AND DUCHESNAY, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research 12* (2011), 2825–2830.

[105] PLATT, J. Sequential minimal optimization: A fast algorithm for training support vector machines. Tech. Rep. MSR-TR-98-14, Microsoft, April 1998.

[106] PUGALENTHI, R., RAJAKUMAR, M., RAMYA, J., AND RAJINIKANTH, V. Evaluation and classification of the brain tumor MRI using machine learning technique. *Journal of Control Engineering and Applied Informatics 21*, 4 (2019), 12–21.

[107] PŁAWIAK, P., ABDAR, M., AND RAJENDRA ACHARYA, U. Application of new deep genetic cascade ensemble of SVM classifiers to predict the Australian credit scoring. *Applied Soft Computing 84* (2019), 105740.

[108] QAYYUM, A., QADIR, J., BILAL, M., AND AL-FUQAHA, A. Secure and robust machine learning for healthcare: A survey. *IEEE Reviews in Biomedical Engineering 14* (2020), 156–180.

[109] Raman, M. G., Somu, N., Kirthivasan, K., Liscano, R., and Sriram, V. S. An efficient intrusion detection system based on hypergraph-genetic algorithm for parameter optimization and feature selection in support vector machine. *Knowledge-Based Systems 134* (2017), 1–12.

[110] Ramirez-Morales, A., Salmon-Gamboa, J. U., Li, J., Sanchez-Reyna, A. G., and Palli-Valappil, A. Boosted support vector machines with genetic selection. *Applied Intelligence 53*, 5 (Mar 2023), 4996–5012.

[111] Sabzevari, M., Martínez-Muñoz, G., and Suárez, A. Building heterogeneous ensembles by pooling homogeneous ensembles. *International Journal of Machine Learning and Cybernetics 13*, 2 (Feb 2022), 551–558.

[112] Seni, G., and Elder, J. *Ensemble methods in data mining: improving accuracy through combining predictions.* Morgan & Claypool Publishers, 2010.

[113] Sesmero, M. P., Iglesias, J. A., Magán, E., Ledezma, A., and Sanchis, A. Impact of the learners diversity and combination method on the generation of heterogeneous classifier ensembles. *Applied Soft Computing 111* (2021), 107689.

[114] Shen, X.-J., Mu, L., Li, Z., Wu, H.-X., Gou, J.-P., and Chen, X. Large-scale support vector machine classification with redundant data reduction. *Neurocomputing 172* (2016), 189–197.

[115] Sivanandam, S., Deepa, S., Sivanandam, S., and Deepa, S. *Genetic algorithms.* Springer, 2008.

[116] Snoek, J., Larochelle, H., and Adams, R. P. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems 25* (2012).

[117] Spencer, R., Thabtah, F., Abdelhamid, N., and Thompson, M. Exploring feature selection and classification methods for predicting heart disease. *Digital health 6* (2020).

[118] Stanley, K. O., and Miikkulainen, R. Competitive coevolution through evolutionary complexification. *Journal of Artificial Intelligence Research 21*, 1 (Feb. 2004), 63–100.

[119] Sudholt, D. *The Benefits of Population Diversity in Evolutionary Algorithms: A Survey of Rigorous Runtime Analyses.* Springer International Publishing, Cham, 2020, pp. 359–404.

[120] Sultan, R., Tamimi, H., and Ashhab, Y. Improving classification performance using genetic programming to evolve string kernels. *The International Arab Journal of Information Technology 16*, 3 (2019), 454–459.

[121] Tan, T. G., Lau, H. K., and Teo, J. Cooperative versus competitive coevolution for pareto multiobjective optimization. In *Bio-Inspired Computational Intelligence and Applications* (Berlin, Heidelberg, 2007), Springer, pp. 63–72.

[122] Tang, J., Alelyani, S., and Liu, H. Feature selection for classification: A review. *Data classification: Algorithms and applications* (2014), 37.

[123] Tang, J., Lim, M. H., and Ong, Y. S. Diversity-adaptive parallel memetic algorithm for solving large scale combinatorial optimization problems. *Soft Computing 11*, 9 (2007), 873–888.

[124] Tao, P., Sun, Z., and Sun, Z. An improved intrusion detection algorithm based on GA and SVM. *IEEE Access 6* (2018), 13624–13631.

[125] Tao, Z., Huiling, L., Wenwen, W., and Xia, Y. GA-SVM based feature selection and parameter optimization in hospitalization expense modeling. *Applied Soft Computing 75* (2019), 323–332.

[126] Tian, D., Zhao, X., and Shi, Z. Support vector machine with mixture of kernels for image classification. In *Proceedings of International Conference on Intelligent Information Processing* (2012), Springer, pp. 68–76.

[127] Ting, K. M., Zhu, Y., and Zhou, Z.-H. Isolation kernel and its effect on SVM. In *Proceedings of The International Conference on Knowledge Discovery & Data Mining* (New York, NY, USA, 2018), pp. 2329–2337.

[128] Wainer, J., and Fonseca, P. How to tune the RBF SVM hyperparameters? An empirical evaluation of 18 search algorithms. *Artificial Intelligence Review 54*, 6 (Aug 2021), 4771–4797.

[129] Wang, D., Qiao, H., Zhang, B., and Wang, M. Online support vector machine based on convex hull vertices selection. *IEEE Transactions on Neural Networks and Learning Systems 24*, 4 (2013), 593–609.

[130] Wang, D., Wang, X., and Lv, S. An overview of end-to-end automatic speech recognition. *Symmetry 11*, 8 (2019), 1018.

[131] Wang, L.-Y., Zhang, J., and Li, H. An improved genetic algorithm for TSP. In *Proceedings of International Conference on Machine Learning and Cybernetics* (2007), vol. 2, IEEE, pp. 925–928.

[132] Waring, J., Lindvall, C., and Umeton, R. Automated machine learning: Review of the state-of-the-art and opportunities for healthcare. *Artificial Intelligence in Medicine 104* (2020), 101822.

[133] Wen, Z., Shi, J., Li, Q., He, B., and Chen, J. ThunderSVM: A fast SVM library on GPUs and CPUs. *Journal of Machine Learning Research 19* (2018), 797–801.

[134] Wolpert, D. H. Stacked generalization. *Neural Networks 5*, 2 (1992), 241–259.

[135] Wolpert, D. H. *The Supervised Learning No-Free-Lunch Theorems.* Springer London, London, 2002, pp. 25–42.

[136] Xiao, J. SVM and k-NN ensemble learning for traffic incident detection. *Physica A: Statistical Mechanics and its Applications 517* (2019), 29–35.

[137] Xu, Z., Aghaabbasi, M., Ali, M., and Macioszek, E. Targeting sustainable transportation development: The support vector machine and the Bayesian Optimization algorithm for classifying household vehicle ownership. *Sustainability 14*, 17 (Sep 2022), 11094.

[138] Yang, L., and Shami, A. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing 415* (2020), 295–316.

[139] Yao, G., Hu, X., and Wang, G. A novel ensemble feature selection method by integrating multiple ranking information combined with an svm ensemble model for enterprise credit risk prediction in the supply chain. *Expert Systems with Applications 200* (2022), 117002.

[140] Zantalis, F., Koulouras, G., Karabetsos, S., and Kandris, D. A review of machine learning and IoT in smart transportation. *Future Internet 11*, 4 (2019), 94.

[141] Zeng, N., Qiu, H., Wang, Z., Liu, W., Zhang, H., and Li, Y. A new switching-delayed-PSO-based optimized SVM algorithm for diagnosis of Alzheimer's disease. *Neurocomputing 320* (2018), 195–202.

[142] Zhang, S., Xu, J., Huang, E., and Chen, C.-H. A new optimal sampling rule for multi-fidelity optimization via ordinal transformation. In *Proceedings of 2016 IEEE International Conference on Automation Science and Engineering* (2016), p. 670–674.

[143] Zhang, W., Hong, B., Liu, W., Ye, J., Cai, D., He, X., and Wang, J. Scaling up sparse support vector machines by simultaneous feature and sample reduction. In *Proceedings of International Conference on Machine Learning* (2017), pp. 4016–4025.

[144] Zhang, W., and King, I. Locating support vectors via β-skeleton technique. In *Proceedings of International Conference on Neural Information Processing* (2002), pp. 1423–1427.

[145] Zhou, J., Qiu, Y., Zhu, S., Armaghani, D. J., Li, C., Nguyen, H., and Yagiz, S. Optimization of support vector machine through the use of metaheuristic algorithms in forecasting TBM advance rate. *Engineering Applications of Artificial Intelligence 97* (2021), 104015.

Acknowledgements

Firstly, I would like to thank my supervisor, Michał Kawulok, for his exceptional guidance and patience throughout this journey. His invaluable insights and continuous support have been instrumental in shaping my research, and I have learned a great deal from all of our discussions. I would also like to extend my sincere thanks to my co-supervisor, Jakub Nalepa, for his countless comments, all of the grammar corrections, and guidance in the scientific community. His support and feedback have been indispensable in helping me achieve my goals.

I cannot thank my wife enough for her unwavering support, encouragement, and belief in me. She has been my pillar of strength, reminding me to take much-needed breaks and providing me with the support I needed to push through the difficult times. Thank you for all the laughs along the way. My heartfelt thanks also go out to my entire family for their support throughout my academic journey. I am grateful for all the help they have provided over the years.

This work has been financially supported by the European Union through the European Social Fund (grant POWR.03.02.00-00-I029) and the National Science Centre (grant 2017/25/B/ST6/00474).

Appendices

A Recursive features elimination

For initial testing, whether optimizing $\mathcal{M}$ and $\boldsymbol{T}$ together with $\boldsymbol{F}$ could further improve the results the recursive feature elimination with cross-validation (RFECV) was exploited. It is a simple yet effective way of finding a refined feature set that can be used with a variety of models.

It works by recursively removing the least important features until the desired number of features is reached. Hence it could be computationally expensive for a large number of features as for each iteration RFE requires training of the machine learning algorithm. In order to decrease the number of training's required, more than one feature could be removed in each iteration. A variant of RFE is presented in Algorithm 10.

At the beginning, a range of steps to test is created, where a step of 10% of features (line 1) is utilized. For each of these steps (starting from full feature set), a 5-fold cross-validation (lines 3-7) is performed. A classifier ($ET_{\boldsymbol{T'}}$) is trained and scored in each fold (lines 5-6). The score from CV is averaged before getting to the next smaller set of features (line 8). The training process (line 10) results in the selection of Z_{best} that yields the best score. The ET is trained one more time for the entire $\boldsymbol{T}$ (line 11) and the least important features are pruned from ET_{best} to obtain Z_{best} number of features, and returned as final $\boldsymbol{F'}$ (line 12).

An extremely randomized tree classifier [51] is used here. The motivation for using this method is its short training time, its non-linearity and the discarding of features based on importance that is natural for this classifier. It may happen that this process discards no features.

Algorithm 10 Recursive Feature Elimination with Cross-Validation (RFECV).

```
1: $\{Z_i\} = \{1.0, 0.9, 0.8, ..., 0.1\}$;                              ▷ Set of steps
2: for all $\{Z_i\}$ do
3:     $\boldsymbol{T'}$, $\Psi \leftarrow$ Split $\boldsymbol{T}$ with $\boldsymbol{F'} = \boldsymbol{F} \times Z_i$ for 5-fold CV .
4:     for all $\boldsymbol{T'}$ do                                    ▷ cross-validation for $\boldsymbol{T}$
5:         $ET_{\boldsymbol{T'}} \leftarrow$ Train classifier on $\boldsymbol{T'}$
6:         $\{Score_{\boldsymbol{T'}}\} \leftarrow$ EVALUATE($ET_{\boldsymbol{T'}}$, $\Psi$)
7:     $\{Score_{\boldsymbol{T'}}\} \leftarrow$ AVERAGE($\{Score_{\boldsymbol{T'}}\}$)
8: $Z_{best} \leftarrow$ Take Step for MAX($\{Score_{\boldsymbol{T'}}\}$)
9: $ET_{best} \leftarrow$ Train classifier for $\boldsymbol{T}$
10: return Take $\boldsymbol{F'}$ from $ET_{best}$ with $Z_{best}$
```

B Statistical tests results

The Friedman test is a non-parametric statistical test used to determine if there are differences between three or more related groups. It is often used as an alternative to the one-way ANOVA when the data is not normally distributed, or the assumption of equal variances is not met [34]. The test involves ranking the observations in each group and calculating a test statistic based on the differences between the ranks. The null hypothesis of the Friedman test is that there are no differences between the populations or treatments being compared. In other words, the null hypothesis is that the medians of the populations or treatments are equal. The alternative hypothesis is that at least one population or treatment median is different from the others. A significant result indicates that there are differences between the groups, but it does not indicate which groups are different from each other. That is why post-hoc analysis needs to be used. The Conover test is a non-parametric pairwise comparison test that is used after a significant Friedman test to determine which groups are significantly different from each other. The test compares all possible pairs of groups using a test statistic that is based on the ranks of the observations. Both tests are non-parametric and based on ranks, making them robust to violations of assumptions about the distribution of the data. The null hypothesis of the Conover post hoc test is that there is no significant difference between the two groups being compared after controlling for multiple comparisons. The alternative hypothesis is that there is a significant difference between the two groups being compared.

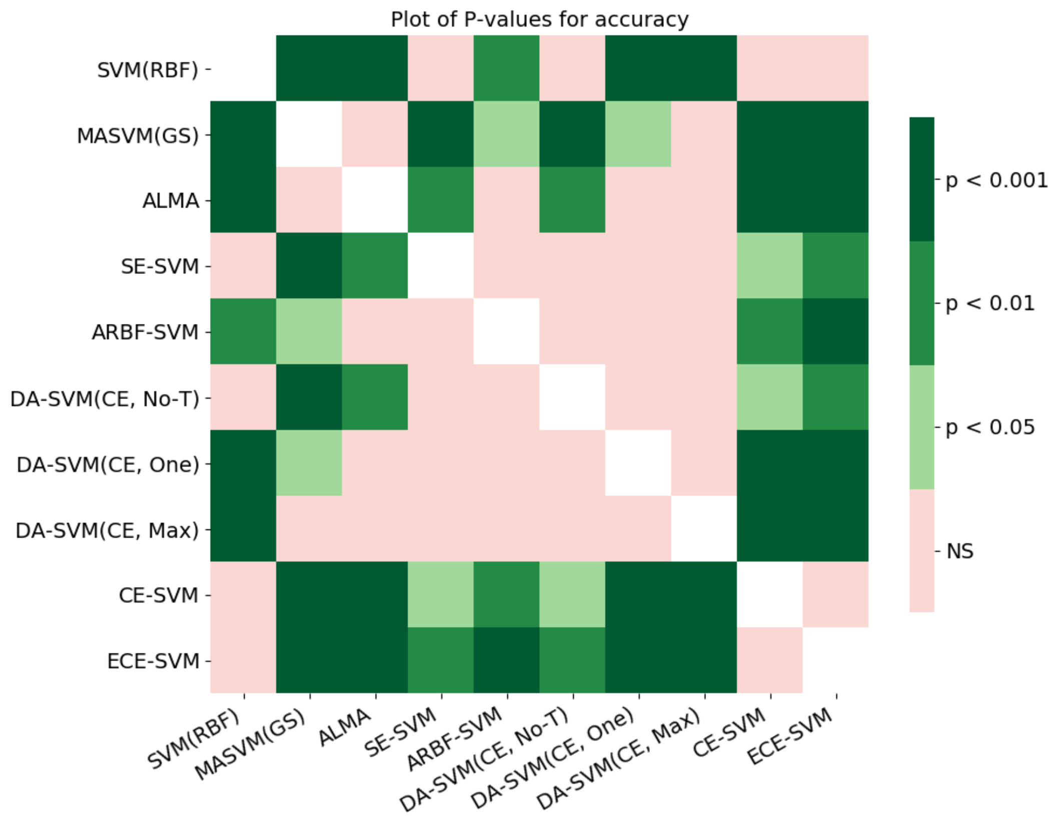


Figure 1: Results of post-hoc Conover analysis performed after Friedman test for 2D datasets on Accuracy metric.

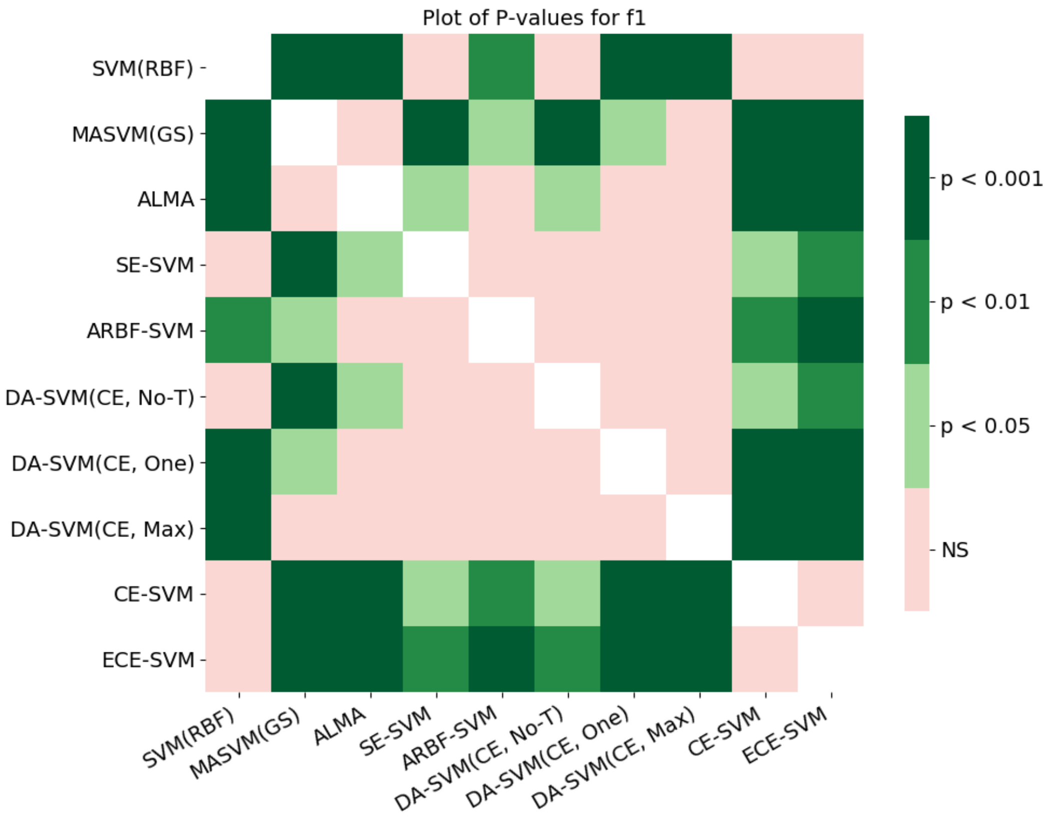


Figure 2: Results of post-hoc Conover analysis performed after Friedman test for 2D datasets on F1 metric.

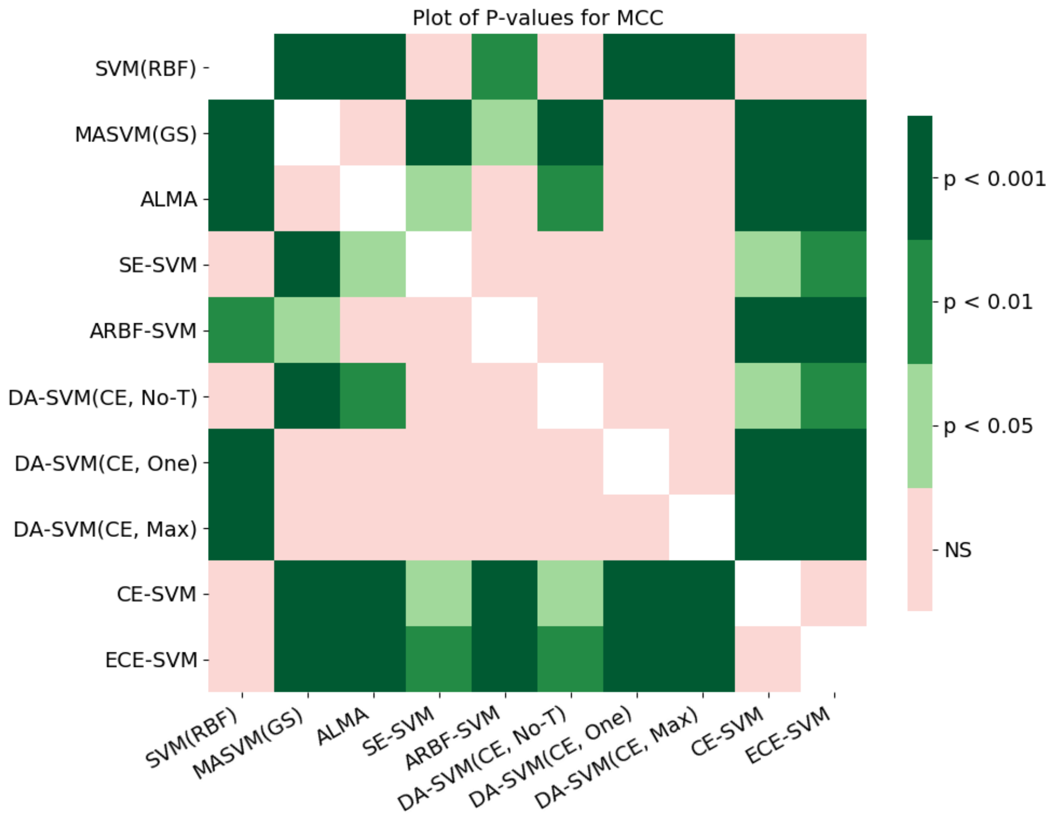


Figure 3: Results of post-hoc Conover analysis performed after Friedman test for 2D datasets on MCC metric.

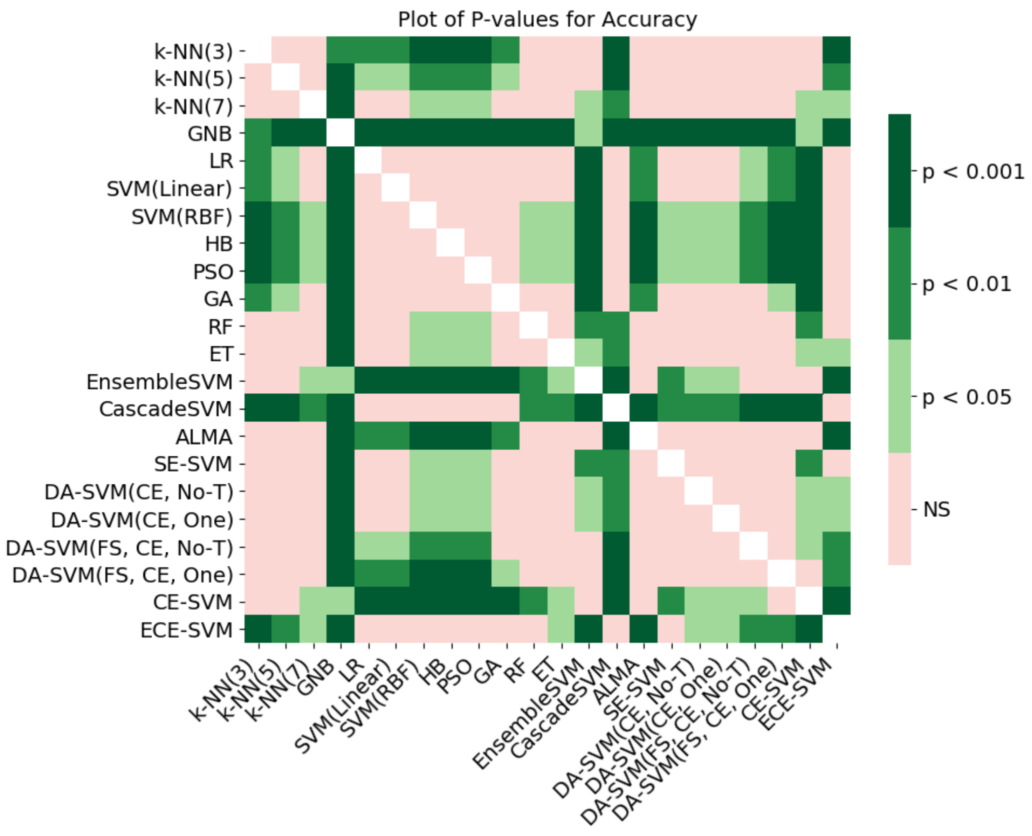


Figure 4: Results of post-hoc Conover analysis performed after Friedman test for benchmark datasets on Accuracy metric.

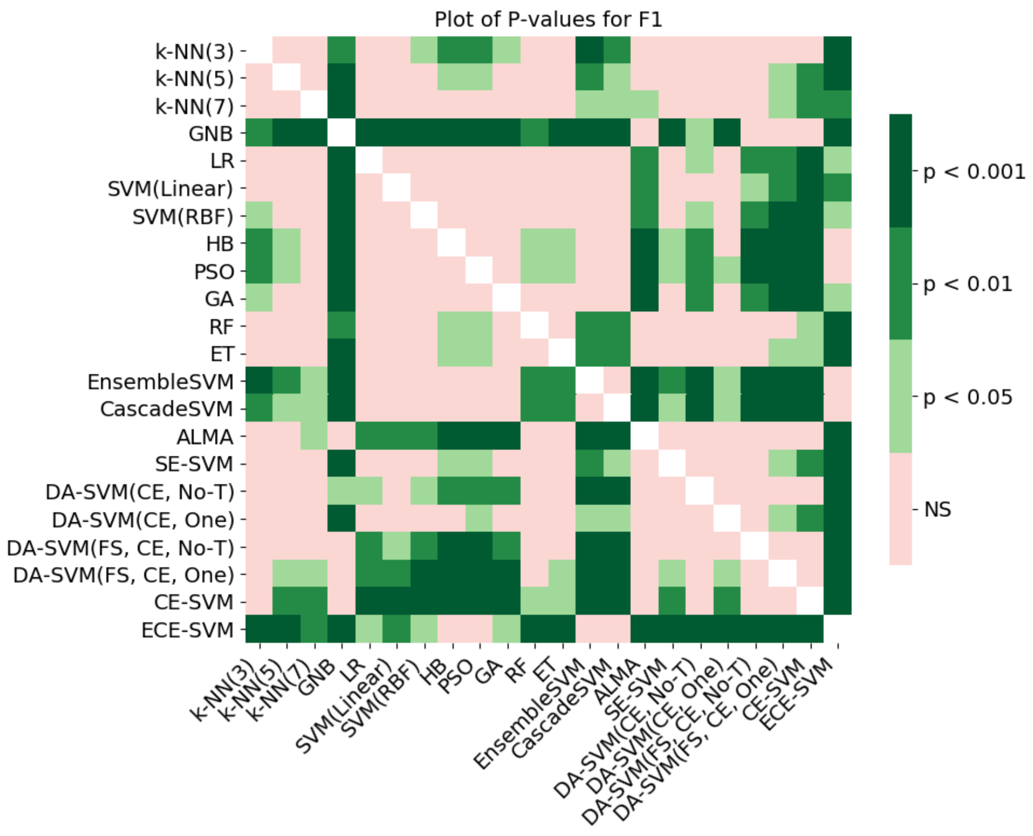


Figure 5: Results of post-hoc Conover analysis performed after Friedman test for benchmark datasets on F1 metric.

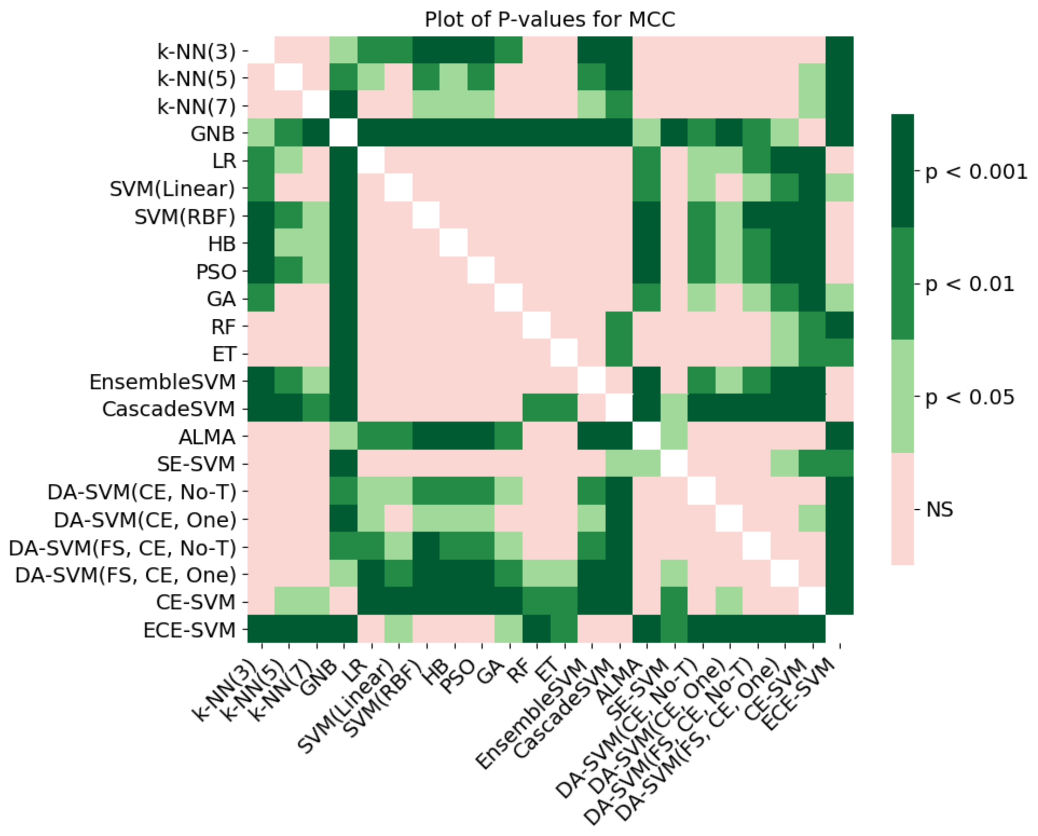


Figure 6: Results of post-hoc Conover analysis performed after Friedman test for benchmark datasets on MCC metric.

List of acronyms

Table 1: List of acronyms in alphabetical order.

Acronym	Description
AI	Artificial intelligence
ANN	Artificial neural networks
AUC	Area under the ROC curve
BO	Bayesian optimization
DNN	Deep neural networks
EA	Evolutionary algorithm
EFS	Evolutionary algorithm for feature selection
ET	Extra tree
FN	False negatives
FP	False positive
GA	Genetic algorithm
GAHP	Genetic algorithm for hyperparameter optimization
GPU	Graphics processing unit
GS	Grid search
HPOs	Hyperparameter optimizations
k-NN	k-nearest neighbors
LGA	Local-global adaptation scheme
MCC	Matthews correlation coefficient
PSO	Particle swarm optimization
RBF	Radial basis function
ROC	Receiver operating characteristic
RS	Random search
SVM	Support vector machine
SV(s)	Support vector(s)
TP	True positive
TN	True negative

List of Figures

List of Tables