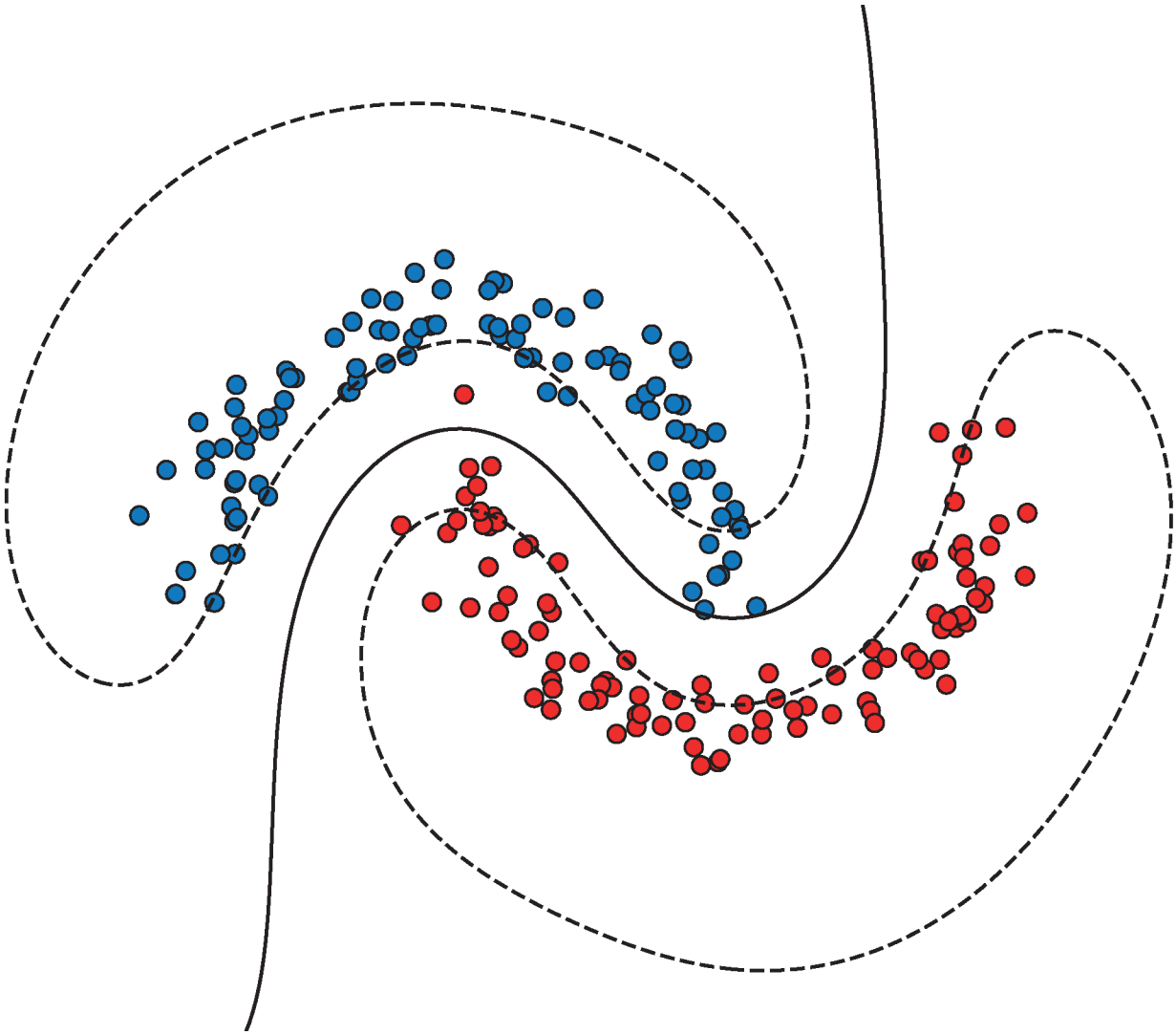
Revisiting SVM Training

Optimizing SVM Hyperparameter tuning using early stopping in the SMO algorithm.

Indy Dekker



Delft University of Technology

TU Delft

Revisiting SVM Training

Optimizing SVM Hyperparameter tuning using early stopping in the SMO algorithm.

by

Indy Dekker

to obtain the degree of Master of Science

at the Delft University of Technology,

to be defended publicly on Monday March 24, 2025 at 11:00 AM.

Student number:	5419018	
Project Duration:	February 2024 - March 2025	
Thesis committee:	M.J.T. Reinders,	TU Delft, Chair
	T.J. Viering	TU Delft, Core Member
	I. B.C.M. Rocha,	TU Delft, Core Member
	O.T. Turan,	Member

An electronic version of this thesis is available at http://repository.tudelft.nl/.

TU Delft

Preface

How did we get here? Is a question that I have frequently asked myself lately and also throughout this study, titled *Revisiting SVM Training: Optimizing SVM Hyperparameter tuning using early stopping in the SMO algorithm."*. As this is the final step in obtaining my Master of Science in Computer Engineering degree at TU Delft, this is a good moment to reflect on this question.

After completing my first degree in Electrical Engineering at Avans Hogeschool, I felt that I had not yet learned what I truly wanted—especially in the fields of Machine Learning and Artificial Intelligence. This was the main reason I decided to pursue further studies at TU Delft. Along the way, I gained technical knowledge and many lessons beyond the coursework. One of the most valuable insights was my growing interest in making complex topics accessible to those without my technical background.

Perhaps this is what led me to focus on Support Vector Machines for my thesis—an algorithm that, while relatively simple compared to modern deep learning methods, remains powerful and widely used. Although I initially had only a basic understanding of SVMs, the expertise of Tom Viering and Taylan Turan quickly brought me up to speed, enabling me to delve deeper and modify the algorithm in meaningful ways. I cannot overstate my gratitude for their patience in guiding me through the many related concepts along the way.

I am also deeply thankful to my thesis committee members, M. J. T. Reinders and I. B. C. M. Rocha, for their time and thoughtful evaluation of my work. Their insights and expertise have been invaluable in shaping this thesis. Together with T. J. Viering and O. T. Turan, they provided the guidance and support that made this journey both rewarding and successful.

My deepest gratitude goes to my friends and family, who stood by me throughout this journey. They had to deal with my research-induced mood swings—the frustrations of days spent chasing a stubborn bug and the endless dinner stories about why a 0.0001-second speedup in my algorithm truly mattered. Despite all this, they stood by me and were there when needed.

A special thank you goes to my girlfriend, whom I met in the final stretch of this project. Her support and encouragement provided the last push I needed to complete this work, even though explaining the so-called ”lines and dots” remains an ongoing task.

Indy Dekker
Barendrecht, March 2025

Abstract

Support Vector Machines (SVMs) are widely used in various domains, with their performance heavily dependent on hyperparameter selection. However, hyperparameter tuning is computationally demanding due to the SVM training complexity, which is at best $O(n^2)$, where n represents the number of training samples. To mitigate this challenge, we propose integrating a validation-based early stopping criterion into the Sequential Minimal Optimization (SMO) algorithm to enhance tuning efficiency.

We evaluate this approach within Random Search and Successive Halving frameworks, aiming to reduce tuning runtime while preserving model performance. We introduce a composite score function to facilitate a balanced assessment of accuracy and efficiency. Our empirical analysis reveals that incorporating early stopping into SMO significantly reduces hyperparameter tuning time under RS but provides limited benefits in successive halving, given its inherent efficiency. Additionally, while dataset characteristics influence the effectiveness of early stopping, we found evidence that dimensionality does not. We also observe that frequent early stopping objective assessments introduce computational overhead, which can offset runtime improvements. Reducing assessment frequency alleviates this issue, but it diminishes the effectiveness of early stopping.

Our findings highlight the potential of early stopping in SMO for optimizing SVM hyperparameter tuning, particularly within random search-based approaches. They also identify trade-offs in assessment frequency and dataset-specific factors.

Contents

Preface i

Abstract ii

1 Introduction 1
1.1 Background . . . 1
1.2 Research Goal . . . 2
1.3 Thesis Outline . . . 3

2 Support Vector Machines 4
2.1 Support Vector Machines . . . 4
2.2 Sequential Minimal Optimization . . . 6
2.3 SMO implementation details . . . 7
2.4 Hyperparameter tuning . . . 8

3 Methodology 11
3.1 SMO with Early Stopping . . . 11
3.2 Effect of early stopping in SMO on SVM HPO. . . . 12
3.3 Effects of data dimensionality on effectiveness of early stopping in HPO. . . . 14
3.4 Effectiveness of early stopping for multiple HPO methods . . . 14

4 Results 16
4.1 Effect of ES in SMO on SVM HPO. . . . 16
4.2 Impact of Datasets on ES in HPO. . . . 19
4.3 Comparing HPO methods . . . 22

5 Discussion 24
5.1 Comparing results with literature . . . 24
5.2 Influence of β setting on results . . . 25
5.3 Statistical analysis . . . 27
5.4 Reducing the ES objective calculation frequency . . . 28
5.5 Limitations and future work . . . 30

6 Conclusion 31

References 32

A SMO Profiling 35
A.1 SMO version comparison . . . 35
A.1.1 Methodology . . . 35
A.1.2 Results . . . 36

B Hinge Loss as an Early Stopping Metric 37
B.1 Hinge loss objective . . . 37
B.2 Results and Discussion . . . 37

1

Introduction

1.1. Background

In an era where the field of machine learning is becoming more and more mainstream through ChatGPT and other similar applications, the inability of humans to understand these models also grows [41]. The problem is that these models are also used in critical areas like medicine or the criminal justice system [32]. Interpretability is still being defined in the machine learning field. While many deep learning methods function as black boxes, offering limited interpretability, some models, such as Support Vector Machines, stand out for their theoretical guarantees on stability and sample complexity, providing a foundation for transparent and robust decision-making in critical applications [39]. Support Vector Machines (SVMs) are supervised learning models that classify data by finding the optimal hyperplane that best separates different classes in a high-dimensional space. They maximize the margin between each class's closest data points (support vectors).

Support Vector Machines (SVMs) remain a prominent tool in contemporary research across various fields [11]. In healthcare, they play a crucial role in disease diagnosis, prognosis prediction, and gene expression analysis, helping to uncover patterns in complex biological data [21]. SVMs are also extensively used in computer vision applications, such as image classification, facial recognition, and object detection, demonstrating their versatility [4, 24, 50]. Additionally, SVMs remain a staple in natural language processing tasks, including text categorization and sentiment analysis, due to their robust performance in high-dimensional spaces [11].

The SVM optimization problem can be formulated and solved in primal and dual forms. While the dual formulation is useful for certain computational advantages, particularly in handling high-dimensional data, the primal approach is often preferred when seeking an approximate solution to the SVM optimization problem. Stochastic gradient descent is commonly used to solve the primal problem, benefiting from faster convergence since it avoids problem decomposition [13, 40].

Several primal solving algorithms, such as Pegasos [40] and Liblinear [17], offer faster convergence times due to their lower runtime complexity (e.g., $O(d)$ for Pegasos, where d is the data dimensionality), making them well-suited for larger datasets. However, a key limitation of these primal methods is their inefficiency when applying non-linear SVM kernels. Even though multiple improvements have been proposed, LibSVM is still considered to be state-of-the-art when used in combination with the subsampling of the data [23].

Dual solvers face challenges such as slow convergence. For instance, LibSVM has a runtime complexity that scales between $O(n^2)$ and $O(n^3)$, where n is the number of training samples [8]. Despite this, LibSVM continues to set the standard by implementing the Sequential Minimal Optimization algorithm [44]. The library receives frequent updates. For example, extensions like warm-starting techniques have been proposed to accelerate convergence [43]. Additionally, GPU-powered dual solvers such as ThunderSVM significantly leverage modern hardware to boost performance [47]. This continued innovation highlights the adaptability and relevance of dual solvers in addressing computational bottlenecks and expanding their applicability in real-world scenarios.

SVMs are a supervised learning technique. The accuracy of the final model depends on the choice of hyperparameters. Figure 1.1 shows the validation error throughout the training of SVMs using different hyperparameter configurations. The figure highlights how the validation accuracy varies with different

hyperparameters, demonstrating the importance of hyperparameter tuning for SVMs. Default methods often used for hyperparameter tuning are random search and grid search [6]. While these methods can be considered brute-force approaches, multiple improvements have been proposed. One example is the successive halving algorithm [27].

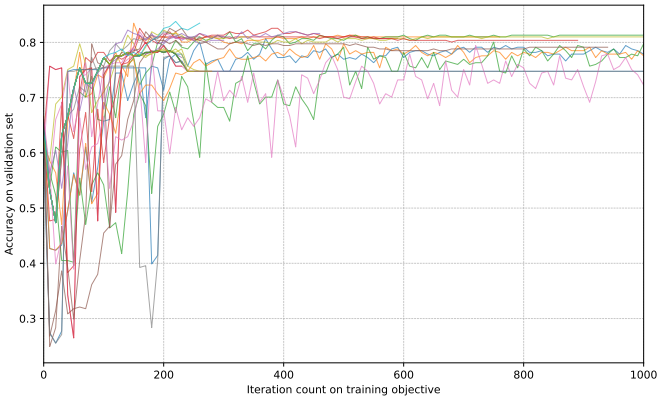


Figure 1.1: Figure illustrating the validation accuracy across various hyperparameter selections for a classification task utilizing SVMs. This figure emphasizes the significant influence of hyperparameter choices on training iterations and convergence behaviour, underscoring the necessity of meticulous hyperparameter tuning for optimal SVM performance.

Hyperparameter optimization (HPO) for Support Vector Machines has been extensively explored in the literature, leading to various algorithmic approaches aimed at improving efficiency [20, 35, 46]. For instance, [20] introduces a multi-objective strategy to optimize hyperparameters, particularly in imbalanced datasets. Given the computational challenges associated with tuning, strategies that integrate adaptive techniques can further enhance the efficiency of the process.

In HPO, we see how network architecture search baselines combine a random search with early stopping; this approach achieves state-of-the-art results in this field [15]. Early stopping is a regularization technique to fight overfitting to the training data. Early stopping is widely applied because of its simple nature and is a superior regularization method, according to [38]. We also see how early stopping is introduced as a dynamic early stopping condition for Random search optimization, where the effectiveness is tested using Support Vector Machines [18].

1.2. Research Goal

Significant progress has been made in SVMs and hyperparameter optimization. Most of the work focuses on designing efficient HPO algorithms that speed up the convergence of the process. However, little attention has been given to incorporating early stopping into the sequential minimal optimization (SMO) algorithm. Figure 1.1 shows us how different hyperparameter combinations influence the validation accuracy throughout training. However, it also shows how, for specific configurations, minimal progress is being made in this objective in the latter stages of training, indicating a potential for early stopping to be beneficial, reducing the overall time to perform such an HPO procedure.

This thesis aims to answer the following question:

"To what extent can early-stopping in the SMO algorithm reduce the runtime of hyperparameter tuning procedures while preserving good performance?"

To answer this question, we defined the following subquestions:

1. Which early stopping configurations yield the best balance of runtime reduction and performance across a range of datasets with diverse characteristics?
2. How does the dimensionality of the dataset impact the effectiveness of early stopping in the SMO algorithm?
3. How do successive halving and random search compare when performing a hyperparameter tuning procedure using the SMO algorithm with early stopping?

To address these questions, we implement the SMO algorithm from scratch in Python. This code will be available as part of my graduation, contributing to future research and reproducibility.

1.3. Thesis Outline

This thesis is organized as follows. Chapter 2 presents the background of this research. Chapter 3 describes the experimental setup used to evaluate the effectiveness of early stopping in hyperparameter tuning for SVMs trained with the SMO algorithm. Chapter 4 details the experimental results. The results are discussed in section 5. Finally, chapter 6 discusses the results and concludes the thesis by providing our main findings.

2

Support Vector Machines

2.1. Support Vector Machines

Support Vector Machines (SVMs) are widely used for classification and regression tasks due to their robust mathematical foundation and strong performance in various applications. At their core, SVMs aim to find a hyperplane that optimally separates two data classes while maximizing the margin between them. Determining the model's parameters is framed as a convex optimization problem, guaranteeing that any local minimum corresponds to the global optimum. This property ensures that the SVM reliably achieves the best possible separation of the data.

The used notations and equations are based on [7]. We define a classification problem where the input vectors are $x \in \mathbb{R}^d$ with corresponding class labels $t_i \in \{-1, +1\}$. Given that $w \in \mathbb{R}^d$ is the weight vector determining the orientation of the hyperplane and b is the bias term, we can define the hyperplane as:

$$\mathbf{w}^T\mathbf{x} + b = 0 \tag{2.1}$$

The points that lie precisely on the margin boundaries, where $t_i(\mathbf{w}^T\mathbf{x}_i + b) = 1$, are known as support vectors. These points are critical because they define the optimal hyperplane. The margin depends entirely on them [11]. Figure 2.1 visually represents a hyperplane for a dataset with two dimensions. Figure 2.1 shows how the support vectors influence the hyperplane.

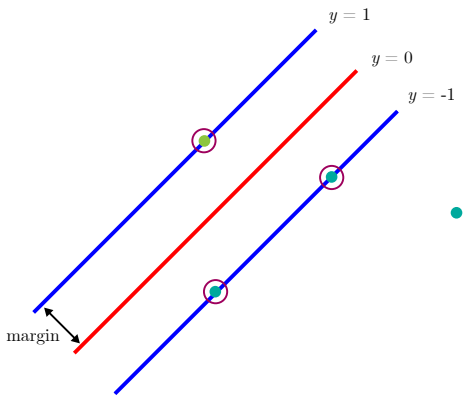


Figure 2.1: Figure showing the optimal hyperplane for a 2D dataset. The margin and support vectors are also indicated.

The primal optimization problem for a Support Vector Machine is given by:

$$\min_w C \sum_{n=1}^{N} \xi_n + \frac{1}{2}\|\mathbf{w}\|^2 \tag{2.2}$$

The first term in the objective function penalizes misclassified points or those within the margins of the decision boundary. The chosen loss function determines the slack variable ξ and assigns a penalty to each training point that violates the margin constraint. The parameter C controls the trade-off between maximizing the margin and minimizing classification errors; a larger value of C results in stricter penalization of misclassified points. The second term in the objective function involves the parameter vector $\mathbf{w}$, which determines the orientation and position of the decision boundary.

SVMs commonly use the hinge loss function to measure a data point's misclassification and margin violations. The hinge loss for vector x_n, given parameter vector **w** is noted as $\xi_n(\mathbf{w}; (\mathbf{x_n}, t_n))$. The term $\langle \mathbf{w}, \mathbf{x_n} \rangle$ represents the inner product between the weight vector w and the feature vector x_n, corresponding to the model's prediction before applying the sign function. We can write the formula of the Hinge Loss as:

$$\xi_n(\mathbf{w}; (\mathbf{x_n}, t_n)) = \max\{0, 1 - t_n \langle \mathbf{w}, \mathbf{x_n} \rangle\}. \tag{2.3}$$

Points on the correct side of the decision boundary with a sufficient margin do not contribute to the loss, while points within the margin do. One problem, however, is that outliers in the data will lead to a considerable loss and thus heavily influence the decision boundary. Due to this, SVMs using the hinge loss are sensitive to noise in the data. This behaviour is illustrated in figure 2.2 since the assigned loss grows linearly with the prediction for data points on the wrong side of the decision boundary.

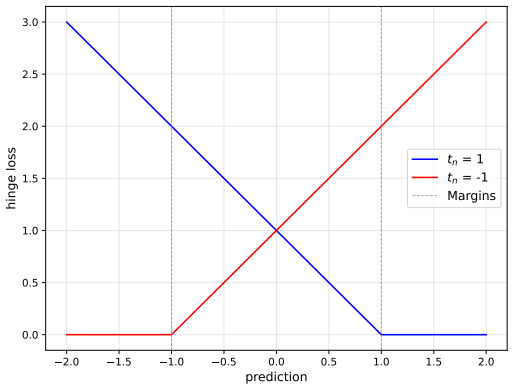


Figure 2.2: The hinge loss function $\xi_n(\mathbf{w}; (\mathbf{x_n}, t_n))$ based on the models prediction $\langle \mathbf{w}, \mathbf{x_n} \rangle$.

The SVM model shown in figure 2.1 has a linear decision boundary. Given equation (2.2), we see that we use the data without any feature-space transformations. However, linear decision boundaries may not effectively capture complex patterns in the data. To address this, we introduce a feature-space transformation denoted by $\phi(\mathbf{x})$. The equation of the hyperplane becomes:

$$\mathbf{w}^T \phi(\mathbf{x}) + b = 0 \tag{2.4}$$

In the primal formulation, solving the optimization problem becomes computationally challenging when the dimension of the feature mapping $\phi(\mathbf{x})$ is large. This is because $\phi(\mathbf{x})$ needs to be entirely stored to solve equation (2.4). The dual formulation leverages the kernel trick to overcome this problem, which allows computations to be performed implicitly by evaluating the kernel function $k(\mathbf{x}_n, \mathbf{x}_m)$ without explicitly storing $\phi(\mathbf{x})$. We rewrite the problem using the Lagrangian dual function that introduces Lagrange multipliers to solve the optimization problem. A detailed explanation of the derivation of this function is provided in [10]. The resulting dual optimization problem becomes a maximizing margin problem, where we maximize:

$$L(\alpha) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \alpha_m t_n t_m K(\mathbf{x}_n, \mathbf{x}_m), \tag{2.5}$$

The optimization in (2.5) is subject to the following constraints:

$$0 \leq \alpha_n \leq C, \tag{2.6}$$

$$\sum_{n=1}^{N} \alpha_n t_n = 0. \tag{2.7}$$

Constraint (2.6) ensures that the Lagrange multipliers remain within a feasible range, with C acting as a regularization parameter that balances the trade-off between maximizing the margin and minimizing classification errors. Constraint (2.7) enforces the separability condition, ensuring that the sum of the weighted Lagrange multipliers aligns with the class labels.

Another advantage of using the dual form optimization is that it inherently exploits the sparsity of the solution, as the Lagrange multipliers α_n are non-zero only for the support vectors. These support vectors are the data points closest to the decision boundary and are crucial for defining the classifier. Finally, the dual formulation reduces the problem to a quadratic programming task, which can be solved efficiently using algorithms such as Sequential Minimal Optimization (SMO) [37].

Once the optimization is complete, the support vectors determine the decision function $f(\mathbf{x})$ for a new input $\mathbf{x}$. Given b is the bias term and $K(\mathbf{x}_n, \mathbf{x})$ the kernel function that captures the similarity between a support vector $\mathbf{x}_n$ and the input $\mathbf{x}$, determining its contribution to the decision function. Then $f(\mathbf{x})$ is given by:

$$f(\mathbf{x}) = \sum_{n} \alpha_n t_n K(\mathbf{x}_n, \mathbf{x}) + b \tag{2.8}$$

2.2. Sequential Minimal Optimization

To solve the SVM problem, we calculate the Lagrange multipliers (α) using the dual formulation (2.5), which results in a quadratic programming (QP) problem [37]. This QP is subject to linear equality and box constraints, where the Karush-Kuhn-Tucker (KKT) conditions ensure optimality by enforcing stationarity, feasibility, and complementary slackness. For all i, the solution satisfies:

$$\begin{aligned} \alpha_i = 0 &\iff t_i f(\mathbf{x_i}) \geq 1, \\ 0 < \alpha_i < C &\iff t_i f(\mathbf{x_i}) = 1, \\ \alpha_i = C &\iff t_i f(\mathbf{x_i}) \leq 1. \end{aligned} \tag{2.9}$$

Decomposition techniques are essential for addressing memory issues when solving the SVM problem, especially for large datasets. Storing the entire kernel matrix with a space complexity of $O(n^2)$ can lead to memory problems, particularly when non-linear kernels are used. To overcome this, methods like Chunking, introduced by Vapnik [45], simplify the optimization by removing kernel-matrix rows and columns corresponding to zero Lagrange multipliers, reducing the problem's size. This allows the larger QP problem to be decomposed into smaller subproblems. SMO is an example of such a decomposition technique, enabling more efficient optimization by solving subproblems with fixed sizes. Other decomposition methods, such as the Osuna algorithm [25] and coordinate descent [49], have also been proposed. However, SMO is one of the most used decomposition techniques.

The SMO algorithm decomposes the SVM quadratic programming (QP) problem into the smallest possible subproblems. Instead of solving the entire QP problem, SMO iteratively selects a pair of multipliers (α_i, α_j) and optimizes them while keeping all other multipliers fixed. This reduces the problem size to a 2D QP optimization, making the computation faster. The process ensures that the Karush-Kuhn-Tucker (KKT) conditions are satisfied at each step, and the solution converges towards the global optimum without needing to store or manipulate the entire kernel matrix at once.

The dual form of the optimization problem imposes two constraints on the two α's to optimize. The first is the box constraint $0 \leq \alpha \leq C$. Secondly the linear constraint $\sum_{i=1}^{n} \alpha_i t_i = 0$. These constraints

ensure that the solution is feasible and satisfies the conditions for optimality. SMO satisfies the linear constraint by focusing on only two variables, α_i and α_j. Any increase in α_i is matched by a proportional decrease in α_j based on the equation $t_i\alpha_i + t_j\alpha_j = c$, where c is a constant. In the next section, we explain how the SMO algorithm is implemented and how it performs these updates.

2.3. SMO implementation details

Now, we will discuss the SMO algorithm in detail. SMO iteratively selects pairs of α values that violate the KKT conditions. To update α_j, we define η as the second derivative of the objective function $L(\alpha)$ with respect to α_i and α_j, given by:

$$\eta = K(x_i, x_j) + K(x_j, x_j) - 2K(x_i, x_j) \tag{2.10}$$

The variable η measures the change in the decision function when updating α_i and α_j simultaneously. Let $E_i = f(x_i) - t_i$ be the error for the ith training sample. Given η and the errors E_i, E_j, the new α_j is computed as:

$$\alpha_j = \alpha_j^{\text{old}} + t_j(E_i - E_j)/\eta \tag{2.11}$$

The update rule adjusts α_j by adding a term proportional to the difference in prediction errors between two training points, scaled by the curvature of the objective function η. This step aims to reduce the error difference while satisfying the optimization constraints. After calculating the new value for α_j we need to clip according to the box constraints enforced by variables $[L, H]$, where $L = \max(0, \alpha_j^{\text{old}} - \alpha_i^{\text{old}})$ and $H = \min(C, C + \alpha_j^{\text{old}} - \alpha_i^{\text{old}})$. Given these bounds we clip α_j as follows:

$$\alpha_j = \begin{cases} H, & \text{if } \alpha_j > H, \\ L, & \text{if } \alpha_j < L, \\ \alpha_j, & \text{otherwise,} \end{cases} \tag{2.12}$$

We define $s = t_i t_j$, we are then able to calculate the value of α_i using the clipped value of α_j:

$$\alpha_i = \alpha_i^{\text{old}} + s(\alpha_j^{\text{old}} - \alpha_j) \tag{2.13}$$

The update for α_i is derived from the constraint that the weighted sum of changes in the Lagrange multipliers must remain zero, so any increase in α_j is balanced by a corresponding decrease in α_i scaled by the product of their labels. This ensures that both multipliers are updated in a coordinated way that preserves the overall constraint of the SVM formulation.

The efficiency of the SMO algorithm depends on the selection of the two alphas for optimization. The original version of the Working set selection heuristic proposed by Platt was not efficient. Thus, multiple improvements have been proposed to speed up the algorithm's convergence. Keerthi et al. [28] improve this by introducing the maximal violating pair approach, which maximizes the objective increase.

Fan et al. [16] enhance working set selection by incorporating second-order information, accelerating convergence. Their key improvement in SMO is using the dual objective gradient (Eq. (2.5)) to select the two α values for optimization. The gradient G_i, representing the sensitivity of $L(\alpha)$ to α_i, is given by:

$$G_i = \frac{\partial L(\alpha)}{\partial \alpha_i} = 1 - y_i \sum_{j=1}^{l} \alpha_j y_j K(x_i, x_j). \tag{2.14}$$

After finishing the main loop of the SMO algorithm, we must calculate the model's bias. We follow the explanation provided in [12]. We introduce a variable ρ, which is $-b$. Based on the KKT conditions we can say that $\rho = t_i G_i$ when there exists α_i, such that $0 < \alpha_i < C$. Note that G_i is the gradient of the dual objective function. For stability, the value of ρ is averaged and is then calculated as follows:

$$\rho = \frac{\sum_{i:0<\alpha_i<C} y_i G_i}{|\{i|0 < \alpha_i < C\}|} \tag{2.15}$$

For an optimal solution, when no Lagrange multiplier satisfies $0 < \alpha_i < C$, the Karush-Kuhn-Tucker (KKT) conditions yield the following constraints on ρ. First, we define:

$$M(\alpha) = -\min\left\{y_i G_i \mid (\alpha_i = 0 \wedge y_i = -1) \vee (\alpha_i = C \wedge y_i = 1)\right\}. \tag{2.16}$$

$$m(\alpha) = -\max\left\{y_i G_i \mid (\alpha_i = 0 \wedge y_i = 1) \vee (\alpha_i = C \wedge y_i = -1)\right\}. \tag{2.17}$$

Thus, the threshold ρ satisfies:

$$M(\alpha) \leq \rho \leq m(\alpha). \tag{2.18}$$

In this case, we take ρ to be the midpoint of the preceding range:

$$\rho = \frac{-M(\alpha) - m(\alpha)}{2}. \tag{2.19}$$

To solve the SVM equation using the SMO algorithm, we need to set the hyperparameter C. We also need to set the stopping tolerance ϵ. In the next section, we cover two approaches to perform an HPO procedure to find hyperparameters that lead to good model performance.

2.4. Hyperparameter tuning

The SVM model contains several hyperparameters that must be specified when training the model. First, the kernel function must be chosen, with standard options being linear, Radial Basis Function (RBF), or polynomial kernels [11]. This research focuses on linear and RBF kernels. The RBF kernel introduces the γ parameter, which controls the kernel width [27]. Smaller γ values produce smoother boundaries, while larger values result in more complex models; this is visualized in figure 2.3. Given this hyperparameter, the RBF kernel is defined in the following way:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2\right) \tag{2.20}$$

The regularization parameter C balances training error minimization and model complexity, with smaller C favouring simplicity and larger C increasing complexity and training time [9]. The tolerance parameter ϵ in SVM optimization defines the threshold for stopping by controlling how closely the solution must satisfy the KKT conditions. Smaller values of ϵ lead to more precise solutions but increase computational cost, while larger values allow faster convergence at the risk of a suboptimal margin.

There are several approaches to optimizing a model's hyperparameters, with random and grid search being among the most well-known. These methods have evolved into more efficient algorithms, such as Successive Halving [27] and Hyperband [31], which aim to reduce computational costs while maintaining effectiveness. This study focuses on random search and Successive Halving, exploring their efficiency and application in hyperparameter optimization.

Random search is a well-known method for tuning hyperparameters, as proposed in [6]. Random search is a hyperparameter optimization method that randomly samples configurations from the search space. Unlike grid search, which exhaustively evaluates all combinations of predefined hyperparameter values, random search treats each parameter as independent and samples values based on a probability distribution. This approach is efficient when only a few hyperparameters significantly influence the model's performance, as it avoids wasting resources on unimportant dimensions. Random search often outperforms grid search in finding high-performing configurations, especially in high-dimensional or complex spaces, by exploring the search space more diversely [6].

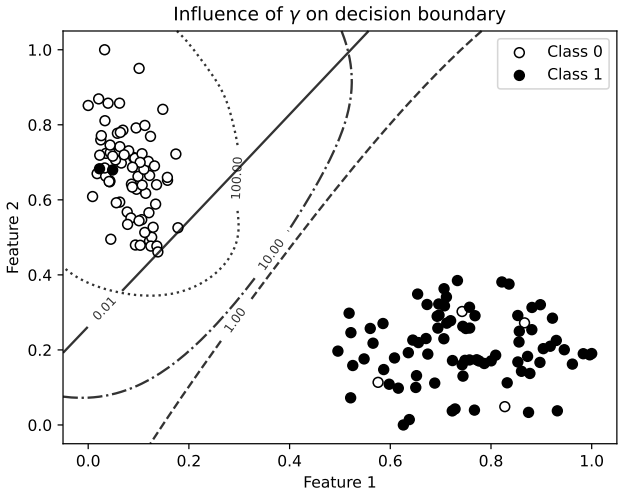


Figure 2.3: Figure showing how γ influences the decision boundary when training SVMs on the same dataset with the same C and ϵ. The values indicate the used value for γ. This Figure illustrates how a higher value of γ leads to a narrower decision boundary.

Successive Halving is an iterative algorithm for hyperparameter optimization that allocates resources progressively to configurations based on their performance [27]. It starts by evaluating many configurations with minimal resources, such as a few training iterations or a small subset of data. After each iteration, the worst-performing configurations are discarded, and the remaining configurations are allocated more resources. This process continues until the allocated budget is exhausted or a single configuration remains; figure 2.4 visualizes this process. By focusing resources on promising candidates early on, Successive Halving efficiently balances exploration and exploitation, making it particularly effective in scenarios with limited computational budgets.

For the use of successive Halving, a budget B needs to be set, for example, a maximum number of iterations or the total number of training samples in a dataset. If we then set the number of hyperparameter configurations we want to try as being k, we can define the number of rounds of the successive halving algorithm as:

$$r = \lfloor \log_2(k) \rfloor + 1 \tag{2.21}$$

If we define k_r as being the remaining models in round r, we can calculate the resources allocated to each configuration as:

$$N_r = \frac{B}{k_r} \tag{2.22}$$

After each round, we eliminate half of the configurations based on their accuracy on a validation dataset. In the last round, we have one remaining that we train on the entire training dataset. This is the best-found model by the hyperparameter tuning procedure.

In addition to deploying efficient HPO strategies like Successive Halving, research also explores combining HPO frameworks such as Random Search with Early Stopping [5, 18]. Early stopping is a regularization technique that halts training once a model's performance on a validation set ceases to improve. ES is a regularization technique that prevents models from overfitting by halting training when the validation loss no longer improves [19, 26].

When applying ES, setting the stopping criteria is crucial. Studies by [5, 38] show that less forgiving stopping rules improve generalization but reduce training time gains. More aggressive ES strategies lead to shorter runtimes but risk performance loss [5]. The aggressiveness of an ES strategy depends on the setting of patience, which defines how many non-progressing steps are allowed. Higher patience results in more forgiving ES, while lower values promote more aggressive ES [26].

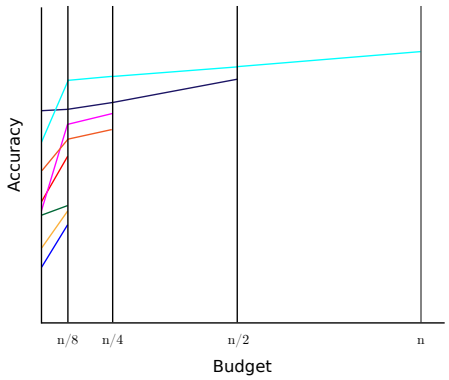


Figure 2.4: Visualization of the successive halving procedure. The X-axis denotes the different rounds of evaluation, while the Y-axis represents the accuracy of the configurations. Half of the configurations are eliminated after each round as the process progresses, illustrating the efficiency of the successive halving strategy in optimization. In this example, $k = 8$.

Note that the ”early stopping” described in the following studies is distinct from the traditional early stopping employed in our work, where training is halted when the validation loss fails to improve. In [5], an ES rule is introduced to reduce computational costs by sequentially evaluating CV folds and stopping early if the remaining folds are unlikely to change the overall outcome. Similarly, [18] applies early stopping directly to the random search process by dynamically assessing whether further evaluations are likely to yield better models, terminating the search if additional evaluations appear unnecessary. Both these studies highlight the potential of ES in HPO.

3

Methodology

This chapter details the methodology used to study the effects of early stopping in SMO from a hyperparameter optimization (HPO) perspective. Section 3.1 details our SMO implementation, including an early stopping condition. Section 3.2 outlines the first experiment, where we examine the impact of early stopping during random searches on multiple datasets, comparing results with and without early stopping. We run this experiment for both RBF and linear kernel SVMs. Section 3.3 focuses on identifying ES configurations from the first experiment that perform well on datasets with varying complexity. These configurations are then tested on synthetic datasets with increasing dimensionality to analyze how data dimensionality influences the effectiveness of early stopping with the best-found configurations. Finally, Section 3.4 presents the third experiment, where the early stopping configurations from Section 3.3 are applied in HPO on new, unseen datasets. This experiment compares the behaviour of early stopping across two HPO methods: successive halving and random search. In these experiments, we solely focus on SVMs trained with RBF kernels.

We normalize all datasets by rescaling features so they all lie between 0 and 1. This step is crucial because it ensures that no single feature dominates the results due to its scale. Normalization significantly improves the performance of Support Vector Machines (SVMs) by enhancing distance calculations between data points [1]. This preprocessing step is essential for achieving better accuracy and reliability in our model predictions.

3.1. SMO with Early Stopping

We based our SMO implementation on the algorithm in [16] and followed its pseudocode in our Python implementation, incorporating runtime optimizations detailed in Appendix A. The algorithm proceeds as follows: first, we select the working set as described in [16]; next, we update the selected α_i and α_j using equations (2.11), (2.12), and (2.13); finally, we update the gradients for all α's according to equation (2.14).

To apply early stopping to this version of the SMO algorithm, we first define an objective. We use the accuracy of the model on a validation dataset for this. Defining $f(x_i)$ as the prediction model for sample x_i and n_{val} as the number of samples in the validation dataset. We can then formulate the objective as follows:

$$O_{val} = \frac{1}{n_{val}} \sum_{i=1}^{n_{val}} 1[f(x_i) = t_i] \tag{3.1}$$

Given the objective function at iteration l of the SMO algorithm, let O_{val}^{l} denote the objective value at iteration l, and O_{val}^{best} represent the best-measured objective value observed so far. Additionally, we introduce two hyperparameters: ϵ_{es}, which specifies the early stopping tolerance, and p, which defines the patience for early stopping. These are two standard hyperparameters associated with early stopping. From now on, when we refer to an early-stopping configuration, we mean a combination of hyperparameters p and ϵ_{es}.

We introduce the variable p_{cur}^{l}, an integer indicating the algorithm's remaining patience at iteration l. On initialization we set $p_{cur}^{0} = p$. Using these definitions, we can express the remaining patience as:

$$p_{cur}^{l} = \begin{cases} p_{cur}^{l-1} - 1 & \text{if } O_{val}^{l} - O_{val}^{best} \leq \epsilon_{es} \\ p & \text{if } O_{val}^{l} - O_{val}^{best} > \epsilon_{es} \end{cases} \tag{3.2}$$

When the progress made by the algorithm is more significant than the set ϵ_{es} value, we reset p_{cur}^{l}. However, when insufficient progress is being made, we decrease its value. When p_{cur}^{l} becomes smaller than 0, we terminate the SMO algorithm due to insufficient progress in the objective.

Listing 1 gives the pseudocode for implementing the SMO algorithm. We must calculate $f(x)$ to evaluate the objective, as given in (2.8). For this calculation, the bias b is used. Thus, we calculate the bias every iteration, as shown in line 8 of listing 1.

Algorithm 1 Implementation of the SMO algorithm with early stopping.

Require: p, ϵ_{es}, `early_stopping_enabled`, `log_enabled`
1: $O_{val}^{best} \leftarrow 0$
2: $p_{cur}^{l} \leftarrow p$
3: **while** `SMO not converged` **and** $l <$ `Max_iter` **and** $p_{cur}^{l} \geq 0$ **do**
4: `Working_Set_Selection()`
5: `Optimize_Alphas()`
6: `Update_Gradient()`
7: `bias` $\leftarrow$ `Calculate_Bias()`
8: **if** `logging_enabled` **then**
9: `Log_Objective_Functions()`
10: **end if**
11: **if** `early_stopping_enabled` **then**
12: **if** $O_{val}^{l} - O_{val}^{best} \leq \epsilon_{es}$ **then**
13: $p_{cur}^{l} = p_{cur}^{l-1} - 1$
14: **else**
15: $p_{cur}^{l} = p$
16: $best \leftarrow l$
17: **end if**
18: $p_{cur}^{l-1} = p_{cur}^{l}$
19: **end if**
20: **end while**

3.2. Effect of early stopping in SMO on SVM HPO.

In this experiment, we investigate if speedups can be achieved for hyperparameter tuning while still finding a good-performance model. To explore this, we perform HPO procedures on multiple datasets with and without ES. We compare the results to see if ES can improve HPO procedures when training SVMs using the SMO algorithm. We rely on 5-fold cross-validation with random search as an HPO procedure. We use this method because a random search is an effective way to explore the hyperparameter space [6], and this form of cross-validation is also applied in [16]. We first perform a random search without early stopping as a reference. We repeat this random search 100 times, using the same configurations for C, ϵ and γ, but for different early stopping configurations. We investigate which early stopping configurations utilize less computational budget and preserve generalization performance.

Figure 3.1 illustrates the experiment setup, where we perform an HPO procedure. We use 5-fold cross-validation following the setup of [16]. The dataset is split into 60% training, 20% validation, and 20% test data. We try 400 different hyperparameter configurations for the reference random search, in line with [29], evaluating the accuracy of each model on the validation set to identify the optimal hyperparameters. After the HPO, we calculate the accuracy of the best model on the test data.

We do not retrain the model using combined validation and training data. If retraining were applied, it would also need to be done for early stopping, which requires reserving a validation set. This would alter the dataset used for retraining, introducing inconsistency.

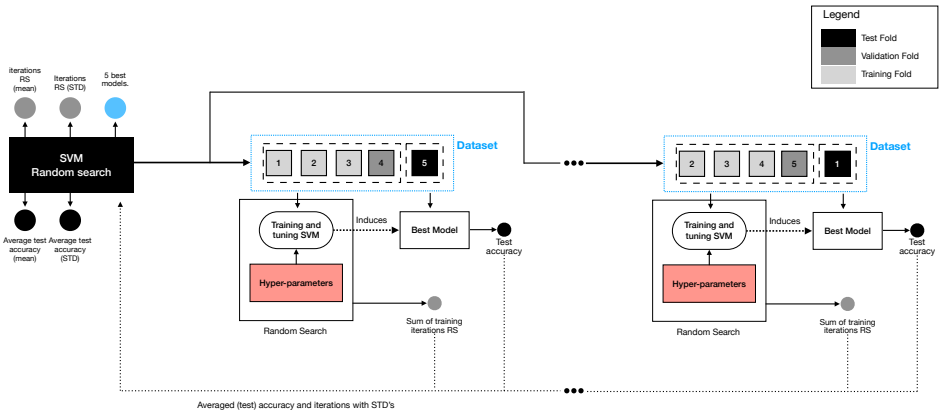


Figure 3.1: Schematic representation of reference random search for tuning hyperparameters of an SVM using stratified 5-fold cross-validation. The hyperparameters are shared between the five runs.

We uniformly draw 400 samples for hyperparameters C, ϵ and γ using the same bounds as in [16], also shown in table 3.1. We also set values for the early stopping hyperparameters p and ϵ_{es}. Table 3.1 also shows the bounds for these parameters. For this experiment, we used accuracy as the early stopping objective.

Table 3.1: Hyperparameter ranges for the random search experiments using linear and RBF kernels. For each hyperparameter, values are uniformly sampled from a continuous distribution within the specified bounds. The range endpoints represent the minimum and maximum values considered during the search.

kernel	$log_2(C)$	$\log_2(\gamma)$	$log_2(\epsilon)$	$log_{10}(p)$	$log_{10}(\epsilon_{es})$
linear	$[-3, 5]$	-	$[-8, -1]$	$[0, 4]$	$[-5, 0]$
RBF	$[-5, 15]$	$[-15, 3]$	$[-8, -1]$	$[0, 4]$	$[-5, 0]$

Our research goal is to study if early stopping can reduce the runtime of an HPO procedure while still finding a model with good performance. Hence, we are dealing with a multi-objective optimization problem. We define a scalarized score function inspired by [22] to evaluate the random search results (accuracy and sum of iterations). We define this function, combining the accuracy and the number of iterations, as follows:

$$S = \beta * \text{acc} + (1 - \beta) * \left(1 - \frac{\text{iter}}{\text{iter}_{\text{ref}}}\right) \tag{3.3}$$

To study the effects of early stopping, for each dataset, we perform a reference random search without early stopping, denoting its score as S_{ref}. We repeat the random search 100 times, using identical configurations of C, ϵ, and γ (for RBF kernels) but varying early stopping configurations. This yields a solution set $T^i = S^i_1, \ldots, S^i_{100}$, where S^i_j indicates the score for the j'th configuration on the i'th dataset, where $j \in \{1, \cdots, 100\}$. To ensure $S^i_j \in T^i$ maintains the accuracy of S^i_{ref}, we set $\beta = 0.995$, allowing for a trade-off when runtime reductions are significant. We deliberately pick a high value for the beta to ensure that the HPO procedure finds a model with good performance, only allowing for a loss of accuracy when it is traded off for a significant reduction in runtime.

Again inspired by [16], we repeat the procedure for six datasets (following [18]), being: *a1a, australian, breast-cancer, diabetes, fourclass, and splice*. Following [34], the a1a and splice datasets are classified as ”high complexity” due to their over 50 features, while the remaining four are ”low complexity”.

3.3. Effects of data dimensionality on effectiveness of early stopping in HPO.

Feature selection optimization influences SVM performance [2]. Given this impact, we examine how data dimensionality affects early stopping (ES) in HPO. Specifically, we test the hypothesis that dimensionality influences ES effectiveness. To do so, we define three ES configurations: one optimized for low-complexity datasets (j_{low}), one for high-complexity datasets (j_{high}), and one performing well across all six datasets (j_{avg}).

To evaluate how dataset dimensionality affects early stopping in HPO, we perform a reference random search using the setup from Section 3.2. For each dataset, a reference experiment (S_{ref}) is conducted. Let i represent one of the seven datasets. for each S^i_{ref}, we use identical samples for the hyperparameters. Each S^i_{ref} is repeated for the three early stopping configurations j_{avg}, j_{low} and j_{high}.

We create a synthetic dataset by generating two Gaussian blobs, where each dimension of the first blob is sampled using $\mu_1 = 0.0$ and $\sigma_1 = 2.0 * \sqrt{d}$, and each dimension of the second blob is sampled using $\mu_2 = 4.0$ and $\sigma_2 = 2.0 * \sqrt{d}$. We let the σ increase with the dimensionality to ensure that the problem does not become linearly separable. We construct seven different datasets with the same specifications, varying only the dimensionality d and the number of samples n. Following [33], we set $n = 10d$. The first dataset has two features, and each subsequent dataset doubles the number of features, culminating in 128 features for the final dataset.

To study the effects of dimensionality, we run HPO procedures for the eight synthetic datasets with an increasing dimensionality and see how this affects the early stopping behaviour.

We use the results from the experiment described in section 3.2 to determine the best early-stopping configurations. We define a new set of results T_{avg} containing the average score of each configuration j across a set of N datasets.

For j_{low}, we only look at the low complexity datasets, which are the *australian, breast-cancer, diabetes, fourclass* datasets. In the case of j_{high}, we only consider the high-complexity datasets *a1a, splice*. Finally, we consider all datasets when determining j_{avg}. We select the configuration that has the maximum average score. We obtain one ES configuration for each of the different complexities.

3.4. Effectiveness of early stopping for multiple HPO methods

While random search is a commonly used hyperparameter tuning method, advancements have been made. For example, Successive Halving shows potential to reduce the runtime of the hyperparameter tuning procedure while still finding good-performing models. To study the effectiveness of ES in HPO, we also use our SMO implementation to perform SH procedures. We compare the results of an RS and an SH procedure to determine how the HPO method influences the effectiveness of ES in SMO. For this experiment, we use the configuration j_{high} as defined in section 3.3.

The configurations are then tested on three new datasets for this experiment. The *w1a* and *german.numer* datasets are also used in [16]. We normalize the data for improved performance. According to our dataset complexity definition, *w1a* is a complex dataset, with 784 and 300 features, respectively, while *german.numer* is a low-complexity dataset with 24 features.

We apply the same bounds for the SVM hyperparameters as defined in section 3.2. For this experiment, we use j_{high} as the early stopping configuration as found in section 3.3. We apply this configuration in both experiments to see how it translates to new unseen data.

As outlined in section 3.2, the same setup is used for studying the differences between HPO procedures. For this experiment, we run it for 100 different samples for each hyperparameter C, ϵ and γ, following the experiments outlined in [27]. Additionally, we perform successive halving procedures alongside the random searches.

The comparison involves two random search approaches (with and without early stopping) and two successive halving variants (with and without early stopping). We adopt the Scikit-learn approach for successive halving, using the entire training dataset as the budget, rather than the unspecified iteration-based budget from [27]. Initially, each configuration uses a fraction of the training data proportional to

the total samples divided by the number of hyperparameter configurations. In subsequent rounds, the number of configurations is halved, and the data allocated to each is doubled until only one model remains.

We evaluate the effectiveness of early stopping in HPO by comparing runtimes with and without early stopping. Instead of using the iterations in the score function, we now use the actual wall-clock times to calculate the scores. This allows us to study the overhead induced by the objective and bias calculations, which is impossible when measuring the runtime in iterations. Additionally, it allows for a better comparison of the differences between successive halving and random searches.

4

Results

This research investigates the benefits of Early Stopping (ES) in the SMO algorithm for reducing the runtime of hyperparameter optimization (HPO) in SVMs while maintaining high accuracy. In chapter 3, we introduced an SMO implementation incorporating ES and defined several experiments to evaluate its effectiveness. This chapter presents the results of these experiments and discusses the findings.

This chapter is structured as follows. Section 4.1 explores the effects of early stopping (ES) in SMO on SVM hyperparameter tuning. Section 4.2 investigates how dataset dimensionality influences ES. Section 4.3 compares SH and RS, both with and without ES.

4.1. Effect of ES in SMO on SVM HPO.

Figures 4.1 and 4.2 show that specific early stopping configurations across the six datasets reduce tuning runtime while maintaining or improving accuracy. However, they also reveal instances where S_j^i accuracy significantly drops due to aggressive early stopping. Aggressive early stopping in this context refers to losing accuracy due to early stopping. These Figures show early stopping configurations that enhance generalization, as seen in the *a1a* dataset. This leads to models with higher test set accuracy, suggesting that the model can overfit training data, with early stopping mitigating this to a minor extent.

We also performed this experiment using the hinge loss on the validation dataset as an early-stopping objective. The results are in the appendix B. The Hinge loss does show similar behaviour compared to using the accuracy.

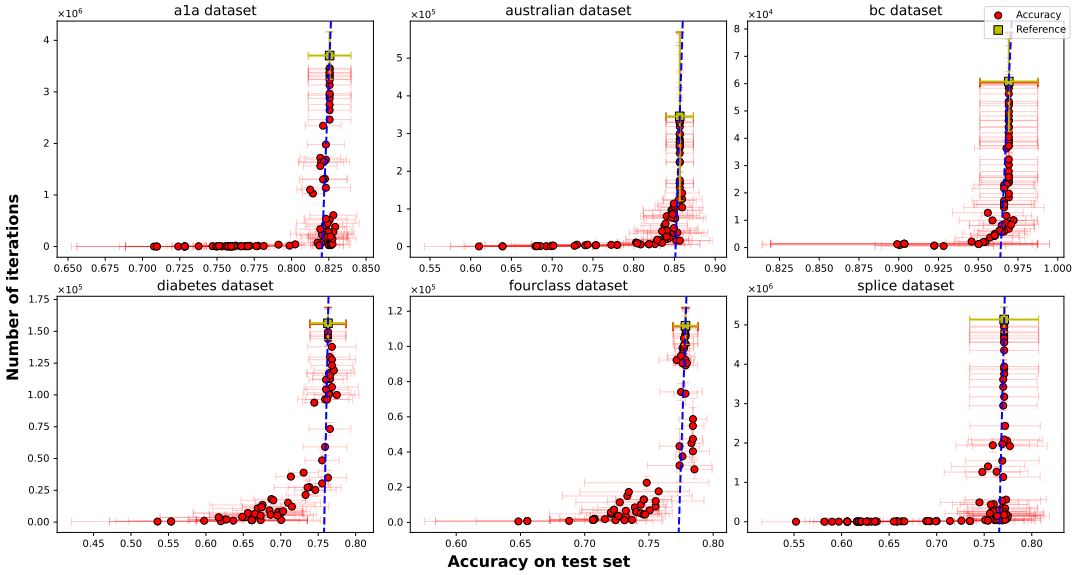


Figure 4.1: Pareto front plots showing total HPO iterations for linear SVMs, with accuracy of the best model on the X-axis. The dashed blue line indicates the score threshold, where points to the right outperform S_{ref}^i. The results demonstrate that ES can enhance performance for all six datasets.

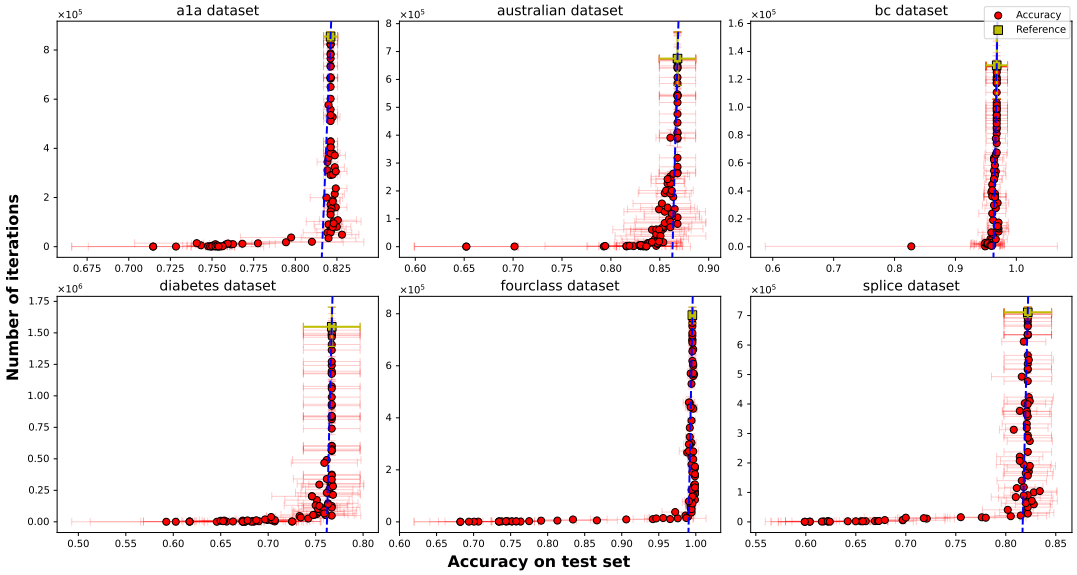


Figure 4.2: Pareto front plots showing total HPO iterations for RBF SVMs, with accuracy of the best model on the X-axis. The dashed blue line indicates the score threshold, where points to the right outperform S^i_{ref}. The results demonstrate that ES can enhance performance for all six datasets.

We use the Wilcoxon test to determine if early stopping benefits the Random Search regarding our defined score function. Following [14], we account for multiple comparisons using Holm's correction. The H_0 hypothesis is that the tested early stopping configuration does not yield gains (in terms of the score) over the six datasets. Rejection of H_0 shows that the specific early stopping configuration j shows significant gains over the N datasets, compared to not using early stopping. We are using a one-tailed test here because we are only interested in the early-stopping configurations that lead to a higher score. The significance level for this test is set to be 0.05.

One reason the Wilcoxon test fails to detect significant results across the datasets can be using Holm's multiple testing correction. This correction becomes stringent since T^i consists of 100 random experiments S^i_j for every dataset i. Additionally, the use of more datasets can also increase the expressive power of the Wilcoxon test.

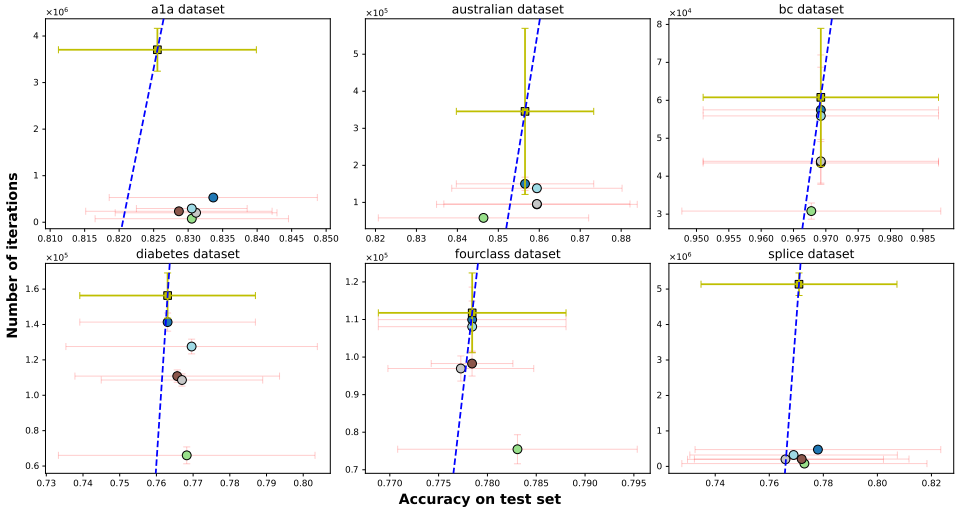


Figure 4.3: Pareto front plot showing S^i_{ref} compared to the five models with the highest average score over the six datasets using the accuracy as an early stopping metric and linear kernels. The plot also shows the score line for $\beta = 0.995$. We see that we do find ES configurations that work well on all six datasets.

The configurations with the highest mean scores for linear and RBF kernel SVMs are shown in Figures 4.3 and 4.4. The results for the random search for linear kernel SVMs, as shown in figure 4.3, show us that the light-blue configuration performs well on all datasets, indicating that this specific configuration generalizes well.

In the experiment for RBF kernels (figure 4.4), we do not see any configurations that score well across all datasets, indicating that the effectiveness of early-stopping configurations strongly depends on the dataset. An example is the light-blue configuration that performs well on all datasets except for the *australian* dataset. Furthermore, we note that of the best-scoring configurations, we see less prevalent results on the *breast-cancer* dataset for both the linear and the RBF kernel results.

Note how there are only four dots in figure 4.3 for the *australian* dataset and figure 4.4 for the *fourclass* dataset. For these instances, the number of iterations and accuracy are precisely equal to the reference, which leads to the point not being visible in the plot because they are behind the reference icon.

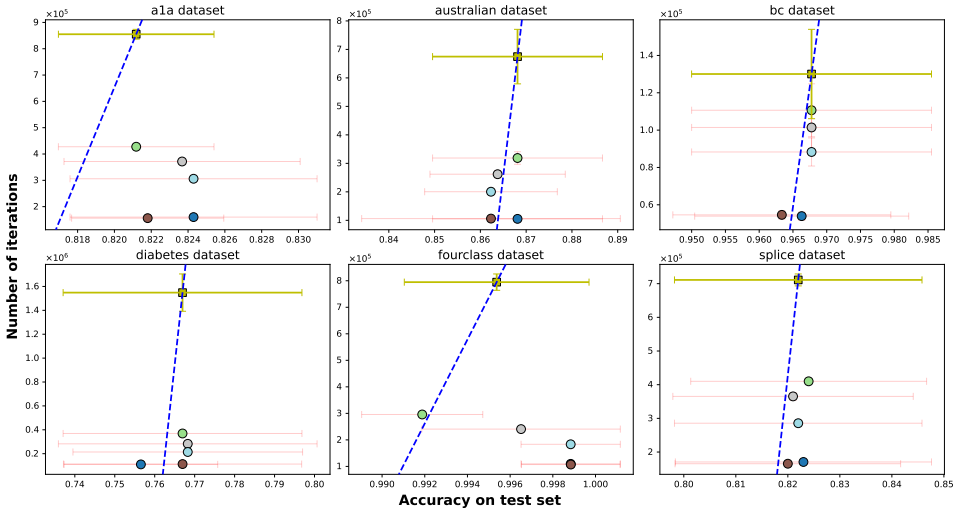


Figure 4.4: Pareto front plot comparing S^i_{ref} with the top five S_j by average score across six datasets, using accuracy as the early stopping metric with RBF kernels. The score line for $\beta = 0.995$ is also shown. The figure highlights how some configurations excel on low-complexity datasets while others perform better on high-complexity data.

Given the results, we see that there are S^i_j for every dataset that outscores the reference without early stopping, indicating that early stopping can be beneficial for decreasing the number of iterations of an HPO procedure while preserving accuracy. However, we cannot detect a single configuration that performs well across all datasets. Hence, it is evident that the effectiveness of early stopping configurations depends on the dataset or that our test setup fails to detect configurations that show promising performance for all datasets.

4.2. Impact of Datasets on ES in HPO.

Figure 4.5 presents the Pareto fronts for three configurations: j_{low}, j_{high}, and j_{avg}, across six datasets. The configurations j_{low} and j_{avg} result in identical settings with $p = 1356$ and $\epsilon_{es} = 0.650653$, while j_{high} uses $p = 109$ and $\epsilon_{es} = 0.000229$.

From the figure, j_{high} exhibits the most aggressive early stopping, significantly reducing iterations across all datasets. However, for some low-complexity datasets (*australian, breast cancer,* and *diabetes*), this comes at the cost of lower accuracy than the reference model. In contrast, the j_{low} / j_{avg} configuration employs less aggressive early stopping, achieving better accuracy across all datasets except the *australian* dataset. This makes it a better choice when prioritizing accuracy over runtime reductions.

The *australian* dataset stands out as an outlier, with none of the tested configurations improving scores. Interestingly, the j_{low} / j_{avg} configuration, which performs well on other low-complexity datasets, fails here. This highlights the significant Influence of dataset characteristics on the effectiveness of early stopping in HPO.

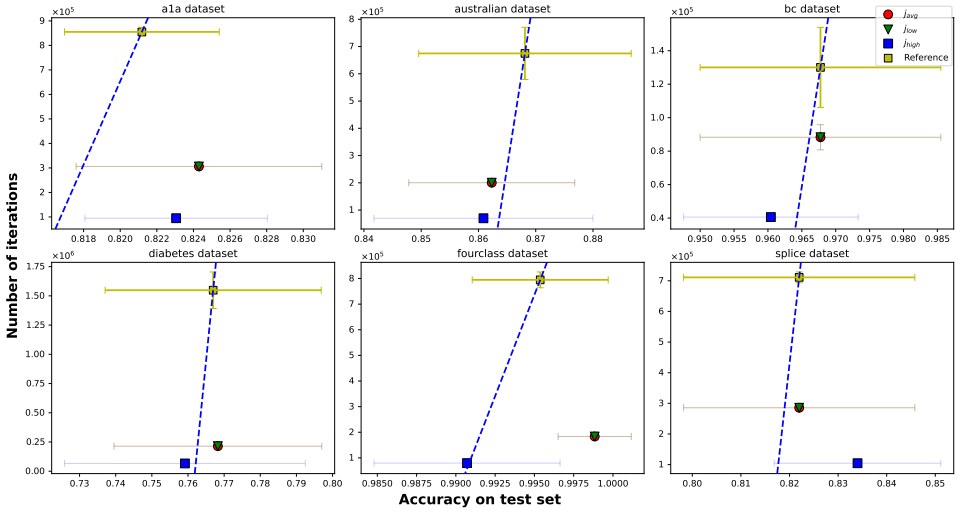


Figure 4.5: Pareto front plot comparing S^i_{ref} with $S^i_{j_{high}}$, $S^i_{j_{low}}$, and $S^i_{j_{avg}}$ over six datasets, using accuracy as the early stopping metric with RBF kernels. The score line for $\beta = 0.995$ is also shown. The plot indicates that j_{low} excels on low-complexity datasets, while j_{high} performs better on high-complexity datasets.

Figure 4.6 compares scores for the two early stopping configurations j_{high} and j_{low} across synthetic datasets, while Figure 4.7 presents their Pareto fronts. The key finding is that dataset dimensionality does not significantly affect early stopping behaviour. j_{low} performs poorly, matching the reference score in lower-dimensional datasets. In contrast, j_{high} shows promising results for all datasets except for d^4 and d^6.

Contrary to expectations, j_{low} improves as dimensionality increases. For lower-dimensional datasets, it provides minimal benefit, but as dimensionality grows, it outperforms the reference score and reduces iteration counts more effectively (Figure 4.7).

Figure 4.7 highlights that j_{high} leads to aggressive early stopping on the synthetic dataset. The results suggest that stronger early stopping is beneficial. While scores in Figure 4.6 remain similar, Figure 4.7 shows a significant reduction in iterations during HPO while still achieving a well-performing model.

Figures 4.6 and 4.7 highlight how our score function is biased towards accuracy gains over iteration reductions. Early stopping appears ineffective in 4.6 as scores remain close. However, 4.7 reveals significant runtime reductions, especially with j_{high}. This discrepancy stems from the choice of β when computing the score. By setting $\beta = 0.995$ to ensure reasonable accuracy, large runtime reductions yield minimal score improvements.

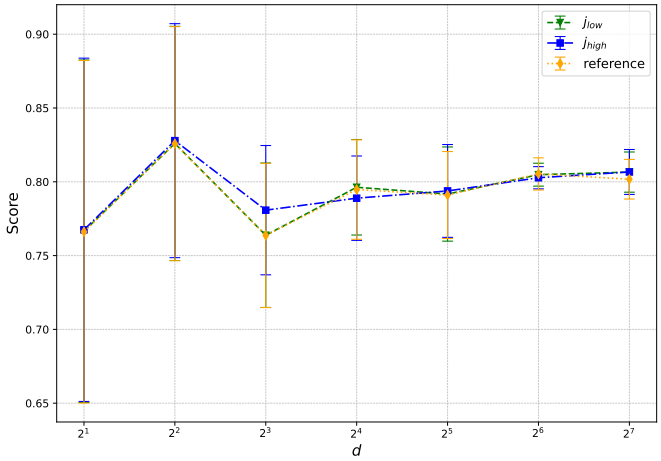


Figure 4.6: S^i_{ref} compared to $S^i_{j_{high}}, S^i_{j_{low}}$ and $S^i_{j_{avg}}$ over 8 datasets with a growing dimension. We do not plot j_{avg} in this plot because it is the same as j_{low}. The two ES configurations show no performance difference as dataset dimensionality increases, highlighting that dimensionality does not impact ES effectiveness in HPO.

This experiment shows that dimensionality influences the effectiveness of early stopping, as j_{low} becomes increasingly effective with higher dataset dimensionality. However, we did not find configurations performing well across low- and high-complexity datasets. Synthetic data does not exhibit the same behaviour as real datasets. As seen in Figure 4.5, j_{low} works well for low-complexity real datasets but not for synthetic data. This suggests that additional dataset characteristics impact the effectiveness of early stopping in HPO.

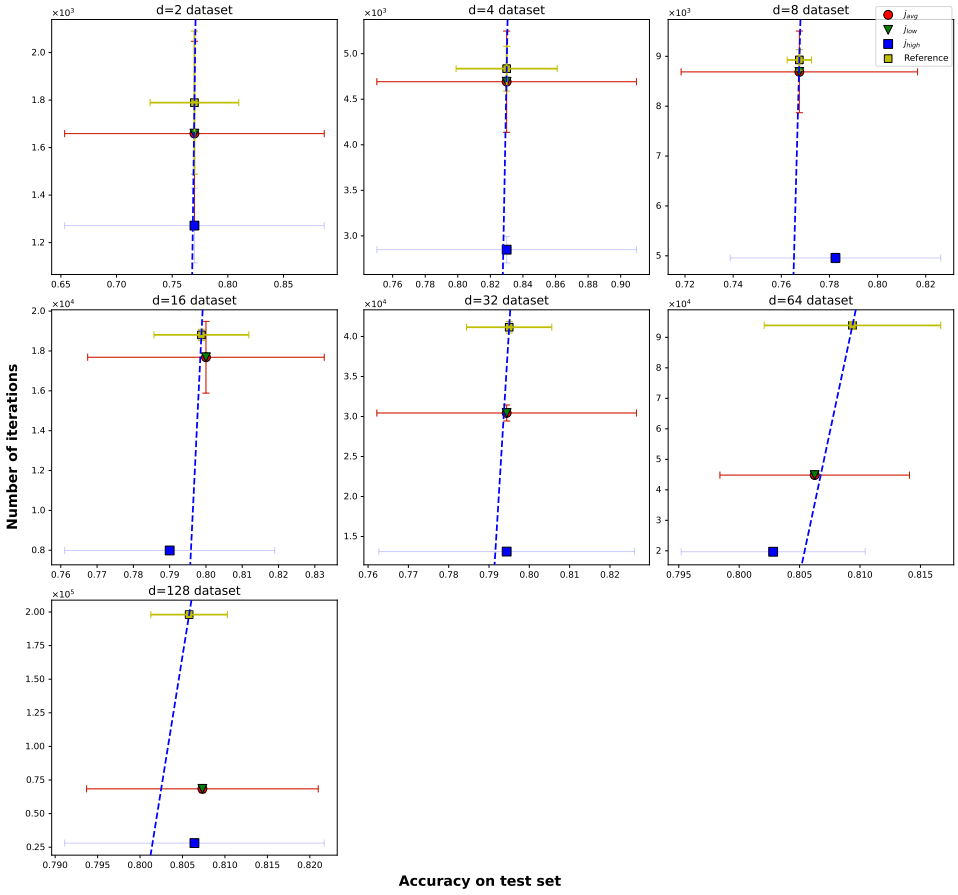


Figure 4.7: S^i_{ref} compared to $S^i_{j_{high}}$, $S^i_{j_{low}}$ and $S^i_{j_{avg}}$ over 7 datasets with a growing dimension. The std for the accuracy is zero for every dataset. The plots show that data dimensionality does not affect the effectiveness of j_{high} and j_{low}, indicating no relationship between dimensionality and ES configuration performance.

4.3. Comparing HPO methods

The results of the four HPO methods on the *w1a* and *german.numer* datasets are shown in Figures 4.8 and 4.9. SH and RS with early stopping do not improve scores on either dataset. However, successive halving consistently runs faster than an RS. Early stopping adds significant overhead, with a only random search on *german.numer*, showing reduced runtime.

When looking at the results for the *w1a* dataset, as shown in figure 4.8, we see that for both HPO methods, the runtime of the experiment increases when applied with early stopping. This shows the overhead induced by the early-stopping calculations, leading to early stopping, which leads to lower scores for both methods.

For successive halving (SH), the best-found model achieves $97.27 \pm 0.17\%$ accuracy without early stopping and $97.33 \pm 0.18\%$ with it, indicating that early stopping does not impact final accuracy. Still, the increased runtime does lead to a lower score. Similarly, for random search (RS), the best-found accuracies are $97.74 \pm 0.30\%$ without early stopping and $97.70 \pm 0.29\%$ with it, showing minimal differences. This suggests that while early stopping affects runtime, it does not significantly alter the accuracy of the best models found.

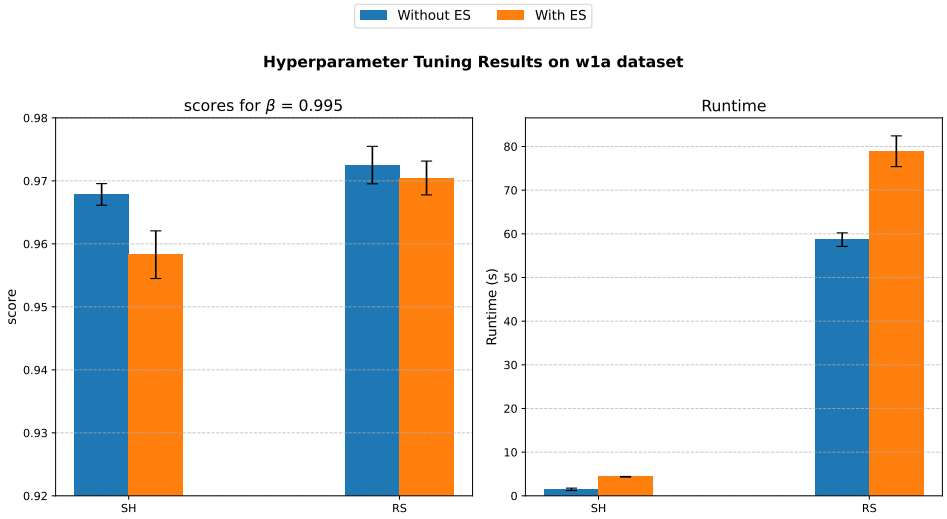


Figure 4.8: Results of running random search and successive halving with five-fold cross-validation on *w1a* dataset using SMO with and without early stopping. The results show that ES is not beneficial for both SH and RS, leading to increased overhead in both HPO methods.

Figure 4.9 shows that for the *german.numer* dataset, we see a score improvement when performing an RS. In the case of SH, we don't see this improvement; for this HPO method, we see how the score drops when applying ES. This shows that ES in this configuration is not beneficial for improving an SH procedure.

Early stopping results in similar accuracies for RS, with $74.94 \pm 1.39\%$ without using ES and $74.98 \pm 0.52\%$ with ES. When using SH, we see a drop in accuracy when using ES from $71.54 \pm 1.28\%$ to $69.42 \pm 2.15\%$. We note that with the setting of β, we penalize loss in accuracy heavily, which explains the decrease in scores for SH.

In this experiment, we observed that early stopping in HPO introduces significant computational overhead, which can lead to longer runtimes. This effect was evident in random search (RS) and successive halving (SH), where the additional calculations required for early stopping delayed the overall process. While early stopping is intended to reduce unnecessary computation, its implementation involves monitoring and decision-making at each step, which can counteract its intended benefits. This finding suggests that applying early stopping without considering its computational cost may not always yield efficiency gains in HPO.

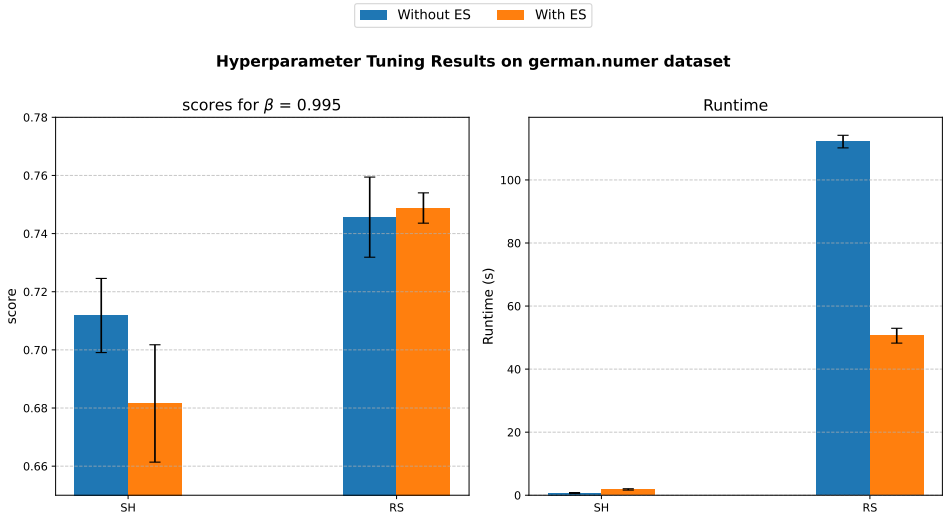


Figure 4.9: Results of running random search and successive halving with five-fold cross-validation on *german.numer* dataset using SMO with and without early stopping. The results show that while ES is not beneficial for SH, it improves scores for RS, demonstrating the benefits of ES.

However, the impact of early stopping is not uniform across all datasets, as dataset characteristics strongly influence its effectiveness. For instance, in the *w1a* dataset, early stopping increased the runtime of RS, likely due to the overhead outweighing the computational savings from stopping earlier. In contrast, for the *german.numer* dataset, early stopping led to a noticeable reduction in runtime, suggesting that it can successfully eliminate unnecessary iterations and improve efficiency in some instances. This disparity highlights that early stopping is not universally beneficial and that its impact depends on factors such as the dataset's structure, the number of support vectors, and the complexity of the hyperparameter search space.

Beyond runtime, we also observed that early stopping led to lower scores for both RS and SH across datasets, but the reasons varied. In the *w1a* dataset, the primary issue was the increased runtime, which delayed the stopping decision and resulted in less effective tuning. In the *german.numer* dataset, the accuracy of the best-found models was lower when early stopping was applied, suggesting that the reduced search time limited the exploration of high-performing configurations. This reinforces the idea that accuracy becomes the dominant factor in determining the final score when a high value of β is used. Therefore, early stopping can reduce runtime under certain conditions and lead to suboptimal model selection if not carefully tuned.

5
Discussion

This chapter analyzes the results presented in Chapter 4. We begin in Section 5.1 by comparing our findings with those in the literature. Section 5.2 examines the impact of the β parameter on the experimental results, which controls the trade-off between accuracy and runtime reduction in our score function. Section 5.3 looks at the Wilcoxon Signed Rank test results to understand why no significant findings emerged. Section 5.4 explores how reducing the ES objective assessment minimizes the computational overhead of our SMO implementation. Finally, Section 5.5 outlines our experimental setup's limitations and suggests future research directions.

5.1. Comparing results with literature

In this section, we compare our experimental results with findings from relevant literature to provide context and validate our conclusions. We examine the effects of early stopping (ES) on model performance, considering insights from [5] and [26] regarding patience and aggressiveness. Statistical analysis methods, including the Friedman test and the preference for the Wilcoxon test, are discussed in light of [18]. We also assess the impact of feature selection on performance, building on the findings of [2], and compare hyperparameter tuning methods, particularly the efficiency of Successive Halving (SH) over Random Search (RS) as outlined by [27]. Lastly, we reflect on the potential of ES, acknowledging that while prior work ([5]) demonstrates its promise, our experiments reveal mixed outcomes.

In [5], the effects of ES on reducing the runtime of a cross-validation procedure are studied. Despite methodological differences, their insights remain relevant. Their study shows that a more forgiving ES strategy yields significant benefits, whereas an aggressive approach can hinder the full exploration of hyperparameter configurations. Patience p is crucial, as higher values require more iterations to complete training [26].

We averaged scores across six datasets to assess ES aggressiveness in the SMO algorithm under RS. We computed the mean patience for configurations yielding positive and negative scores. Positive-scoring configurations had a mean p of 19122, while negative ones averaged 280, aligning with findings in [5].

To compare our assessment of the results found in Section 4.1, we refer to the work of [18]. This study introduces a dynamic stopping condition tailored for a random search to enhance efficiency while maintaining competitive performance. The authors systematically evaluate multiple variants of their proposed algorithm, comparing them against a range of established optimization techniques. Our experiments differ by applying early stopping (ES) to terminate model training. In contrast, their study applies ES directly to the random search process, assessing whether an additional round of random search will yield a better-performing model.

[18] demonstrates that a dynamic early stopping condition in a random search maintains accuracy while significantly reducing runtime—aligning with our findings in Section 4.1 on the benefits of ES in SVM HPO.

The key difference is that [18] uses an RS-based stopping condition, while we propose an alternative SMO algorithm. They apply the Friedman test with the Iman-Davenport statistic. We used the Wilcoxon signed rank test with Holm's correction, following [14]. Whe Wilcoxon test is deemed superior over the

Friedman test as discussed in [3]. While [18] reports significant gains, our statistical setup does not yield the same outcome. Another distinction is that we test 100 configurations, whereas [18] evaluates only four algorithms, affecting the resulting p-values under Holm's correction.

We compare the results from Section 4.1 with those of [2]. While the objectives of the two studies differ, [2] shows that feature reduction or optimization improves model performance. Our study examined the impact of dimensionality on HPO with ES using datasets with varying dimensions. Despite the different methodologies, we expected similar results, but this was not the case.

Our study focuses on the impact of dimensionality on ES in SMO within the context of HPO, whereas [2] investigates the effects of dimensionality on SVM performance. A key methodological difference is that we construct seven datasets where every feature is informative. In contrast, [2]—along with other related work [42, 48]—explores feature reduction by eliminating non-informative features.

The main discrepancy between our findings (as presented in Section 4.2) and those of [2] is that, in our case, the reference score remains within a similar range across all datasets. We initially expected that increasing the number of features would enable RS to find better models, leading to higher scores. However, this was not observed, suggesting that dimensionality alone does not inherently affect performance. The key takeaway is that performance improvements stem from feature selection and optimization, as demonstrated in [2].

We compare the results in section 4.3 with those of [27]. Our research shows that SH outperforms RS on the *w1a* and the *german.numer* datasets. Additionally, both RS and SH found similar accuracies in both experiments. These findings correspond with the results found in [27], where SH is faster on all of the datasets used for that experiment.

In the experiment from section 4.3, we evaluated runtime and found that ES in HPO does not always provide benefits. For the *w1a* dataset, ES unexpectedly increased runtime. However, for the *german.numer* dataset, ES improved efficiency by reducing the runtime of the RS procedure.

5.2. Influence of β setting on results

This section explores how our score function's β parameter influences experiment results. The β value controls the trade-off between prioritizing accuracy and minimizing iterations in HPO procedures, with zero focusing solely on iteration reduction and one on model accuracy. We analyze the impact of different β settings on the results in sections 4.1 and 4.2.

We analyze the top-scoring configurations across the six datasets from section 4.1 for different β values. Figures 5.1 and 5.2 display these configurations. The dot colours represent the best configuration for the score threshold in the same color. As β decreases, the number of iterations drops, showing more aggressive ES. This aligns with the shift in priority from accuracy to iteration reduction when lowering β.

Configurations found for $\beta = 0.8$ (orange) and $\beta = 0.95$ (green) show similar performance, as seen in Figures 5.1 and 5.2. This is especially evident with RBF kernels (Figure 5.2), where the dots for all six datasets are closely clustered.

Table 5.1 lists the patience p and ES tolerance ϵ_{es} values for the configurations in Figures 5.1 and 5.2. For both linear and RBF kernels, patience increases with higher β, indicating that prioritizing accuracy requires a more forgiving ES strategy, achieved by raising p. This finding aligns with [26], which suggests that higher patience results in less aggressive ES.

The key insight is that $\beta = 0.995$ results in a best configuration with forgiving ES behavior, supporting our accuracy-focused goal. However, Figures 5.1 and 5.2 show that even with lower β values ($[0.8, 0.95]$), the best configurations still achieve good accuracy across all datasets for both linear and RBF kernels.

As a conclusion of experiment 1 (section 4.1), we state that there is a difference in the effectiveness of ES configurations for low- and high-complexity datasets. We studied these effects in section 4.2 and did not find significant differences between ES configurations selected for low complexity datasets (j_{low}) and high complexity datasets (j_{high}), which becomes apparent in Figure 4.6.

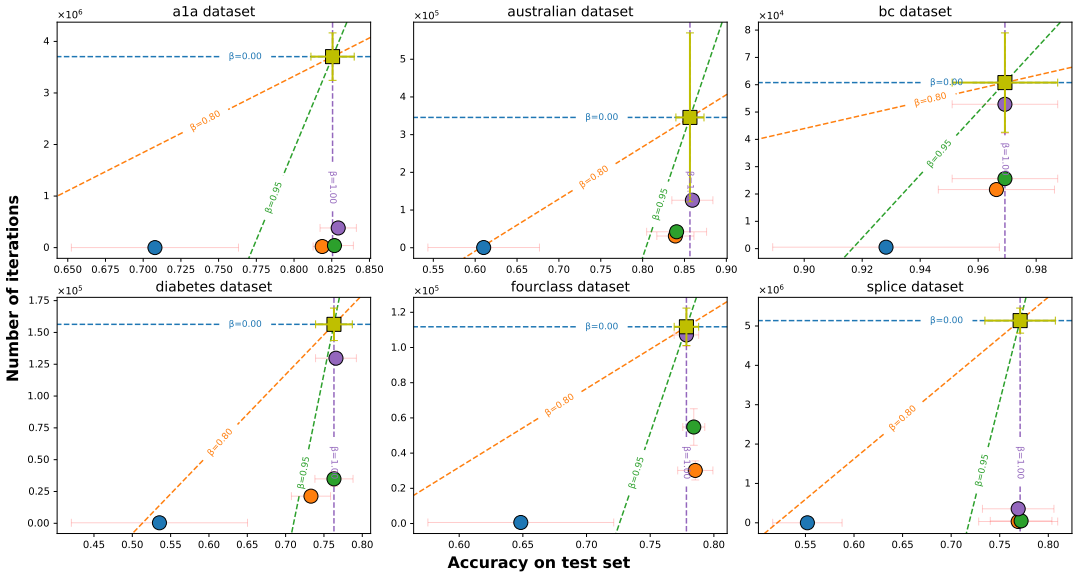


Figure 5.1: Pareto front plot comparing S^i_{ref} to the best-scoring S_j for four values of β in linear-kernel SVMs. The colour of each configuration in the Pareto front corresponds to a score threshold line of the same colour. The plots show that as β decreases, the best ES configuration becomes more aggressive.

Table 5.1: ES configurations j with on average best scores for different values of β for both linear and RBF kernel SVMs.

	linear		*RBF*	
β	p	ϵ_{es}	p	ϵ_{es}
0.0	1	0.061308	1	0.406918
0.8	36	0.000023	44	0.000145
0.95	54	0.000020	114	0.132280
0.995	493	0.004117	1356	0.650653
1.0	493	0.004117	10987	0.017214

To analyze the effects of β on the best-found configurations, we examine the configurations identified for j_{high}, j_{low}, and j_{avg} across multiple β settings. These configurations are selected using the same strategy described in Section 3.3. To evaluate their impact, we report the wins and losses for the best configurations corresponding to the $beta$'s used earlier. This analysis is conducted separately for high- and low-complexity datasets and across all six datasets, following the setup outlined in Section 3.3.

Table 5.2 presents the wins and losses for the identified configurations across different β values. As β decreases—placing greater emphasis on runtime reduction—we observe that all three configurations achieve wins across all datasets. These findings suggest that ES effectiveness differences only emerge when accuracy is strictly enforced ($\beta = 0.995$). The results in Section 4.2 further indicate that this behaviour is dataset-specific, as we could not replicate it using synthetic datasets.

Table 5.2: The number of wins on the three dataset subsets for j_{low}, j_{high}, and j_{avg} using different values of β. A win is defined as $S^i_j > S^i_{ref}$.

	j_{low}			j_{high}			j_{avg}		
β	*low*	*high*	*avg*	*low*	*high*	*avg*	*low*	*high*	*avg*
0.0	4	2	6	4	2	6	4	2	6
0.8	4	2	6	4	2	6	4	2	6
0.95	4	2	6	4	2	6	4	2	6
0.995	3	2	5	0	2	2	3	2	5
1.0	1	0	1	0	2	2	1	0	1

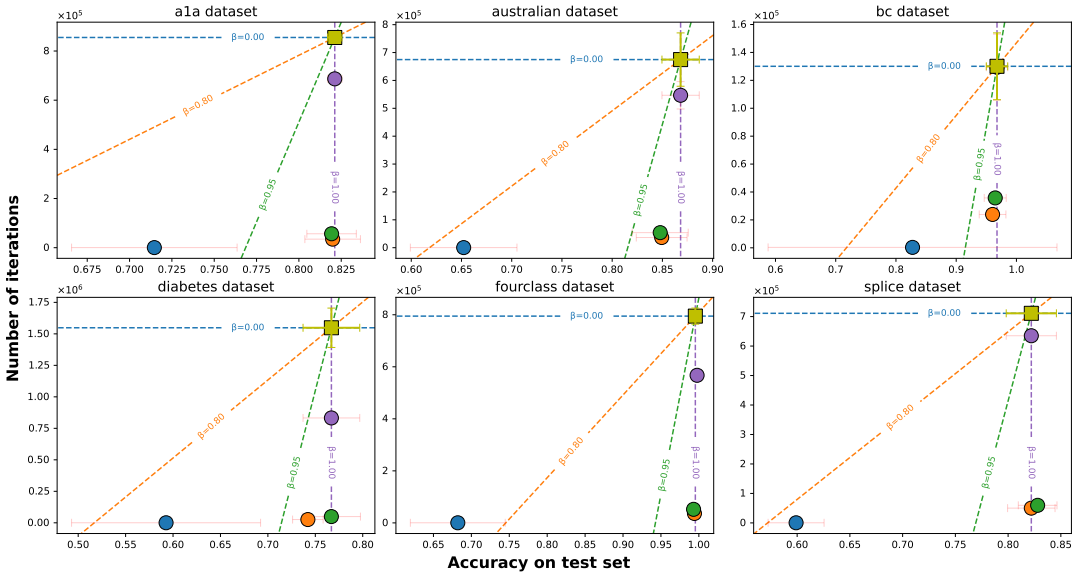


Figure 5.2: Pareto front plot comparing S^i_{ref} to the best-scoring S_j for four values of β in RBF-kernel SVMs. The colour of each configuration in the Pareto front corresponds to a score threshold line of the same colour. The plots show that as β decreases, the best ES configuration becomes more aggressive.

The results in Table 5.2 indicate that dataset dimensionality does not impact the effectiveness of ES in HPO. Furthermore, the differences observed in Section 4.1 can be attributed to the emphasis on maintaining accuracy combined with dataset-specific characteristics. When allowing for a more lenient accuracy drop, no differences emerged in the effectiveness of ES configurations between low- and high-complexity datasets. This aligns with the findings from synthetic datasets, where no significant differences were observed between j_{high}, j_{low}, and j_{avg} for datasets with an increasing dimension.

5.3. Statistical analysis

This section examines the Wilcoxon test with Holm's correction from section 4.1. None of the null hypotheses were rejected for linear and RBF kernels, indicating no significant gains across configurations. However, figure 4.3 shows configurations with positive scores for all six datasets, which we expected the Wilcoxon test to identify as significant. To investigate, we analyze the Wilcoxon test results in detail.

First, we count the wins and losses for each configuration j across the six datasets in this experiment. A win is defined as $S^i_j > S^i_{ref}$, meaning that configuration j achieves a higher score than the reference for dataset i. For ties, we follow the procedure described in the Sign test by [14], where ties are split evenly between wins and losses. Figure 5.3 displays the number of wins and losses for all 100 ES configurations j across the six datasets. The figure reveals that for RBF kernel SVMs, more than 20 configurations have six wins, indicating that these configurations outperform the reference on all six datasets.

Figure 5.3 shows configurations j that improve performance on all six datasets, indicating strong Wilcoxon test results. Achieving wins on all six datasets yields the maximum Wilcoxon score, w_{stat}, calculated as:

$$w^{max}_{stat} = \sum_{i=1}^{n} \frac{(n+1)n}{2} \tag{5.1}$$

For $n = 6$, this results in $w^{max}_{stat} = 21$. However, statistical significance is not reached due to Holm's correction. With 100 tested configurations, the highest possible p-value for $w_{stat} = 21$ is 0.015625. After applying Holm's correction ($p_{adj} = 100 \cdot p = 1.5625$), the adjusted p-value remains above the $\alpha = 0.05$ threshold, blocking statistical significance.

The number of trials limits our current setup for assessing statistical significance. Even in the ideal scenario where a configuration wins on all six datasets, it fails the significance test. To address this, we

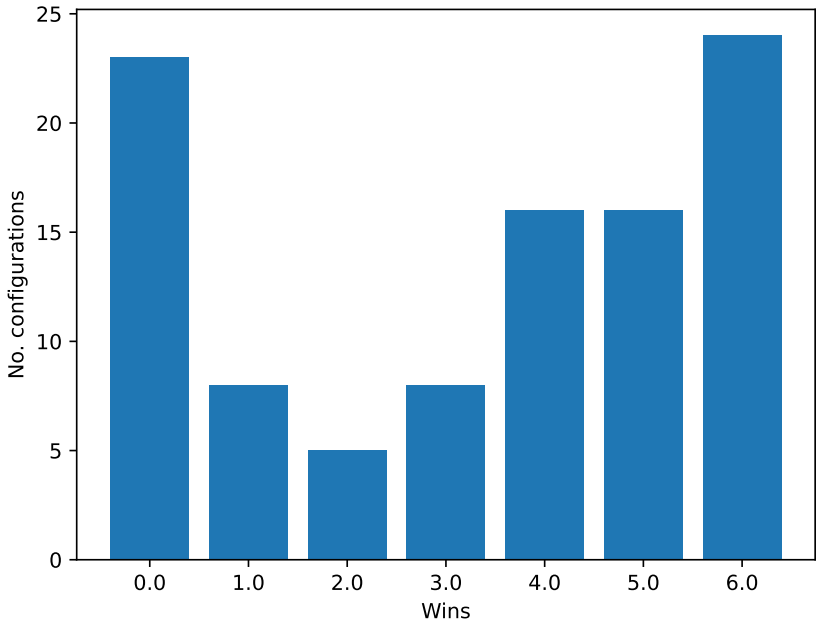


Figure 5.3: Histogram showing the number of wins for each ES configuration j on the six datasets for SVMs trained with RBF kernels. A win is defined as $S_j^i > S_{ref}^i$, and the scores are calculated using $\beta = 0.995$. The figure shows how multiple configurations have wins on all six datasets.

could either increase the number of datasets, raising w_{stat}^{max} and enabling smaller p-values and adjusted p-values (p_{adj}), or reduce the number of tested configurations j, easing the multiple testing correction.

5.4. Reducing the ES objective calculation frequency

In section 4.3, we observed that ES introduces significant overhead due to frequent objective assessments. To address this, we conduct an additional experiment where we reduce the frequency of these assessments. This section examines how this adjustment impacts the results from section 4.3.

In our SMO implementation, we introduce a new hyperparameter, r, determining the number of iterations, after which the ES objective is assessed. For instance, setting $r = 10$ means the ES objective is evaluated every ten iterations.

To identify a suitable ES configuration, j_r, we follow the setup described in section 3.2, but instead of testing 100 configurations, we expand the search to 1000 candidates. The configuration j_r is selected as the one that achieves the highest average score across the six datasets.

To assess whether introducing r improves the results from section 4.3, we repeat the experiment using j_r as the ES configuration.

As a result of trying different ES configurations, we find that j_r has a $p = 1$, $\epsilon_{es} = 0.00012697$ and $r = 768$. Using this configuration when running an SH and RS procedure on the *w1a* and *german.numer* dataset leads to the results found in figures 5.4 and 5.5.

For the *w1a* dataset (Figure 5.4), applying ES does not improve accuracy for either SH or RS, with both methods yielding similar results. SH achieves $97.39 \pm 0.21\%$ without ES and $97.40 \pm 0.23\%$ with ES, while RS reaches $97.74 \pm 0.30\%$ in both cases. This negligible impact on accuracy is primarily due to runtime differences—Figure 5.4 shows that ES increases runtime for both methods. While SH maintains a similar wall-clock time, RS incurs additional overhead from ES calculations, even though the check frequency is significantly reduced ($r = 768$).

For the *German.numer* dataset (figure 5.5), ES improves scores for both SH and RS, unlike in figure

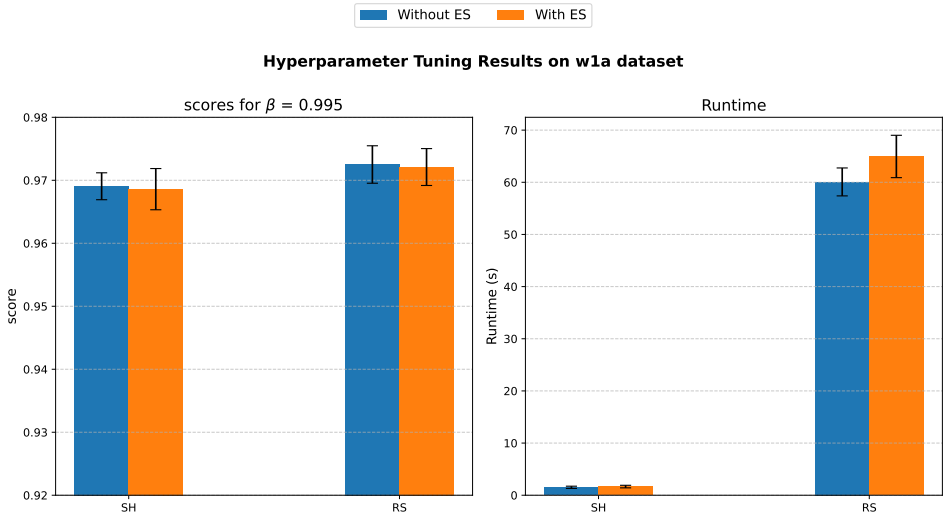


Figure 5.4: Results of running random search and successive halving with five-fold cross-validation on *w1a* dataset using SMO with and without early stopping. When performing fewer ES objective assessments, ES provides no advantage for SH or RS, suggesting that reducing the assessment frequency does not enhance performance.

5.4. SH achieves $71.40 \pm 1.31\%$ without ES and $71.76 \pm 0.40\%$ with ES. RS reaches $74.94 \pm 1.39\%$ without ES and $74.86 \pm 1.45\%$ with ES. The SH score improves due to a slight accuracy gain, favoured by our β setting, while for RS, the runtime reduction outweighs the minor accuracy drop, leading to a higher score.

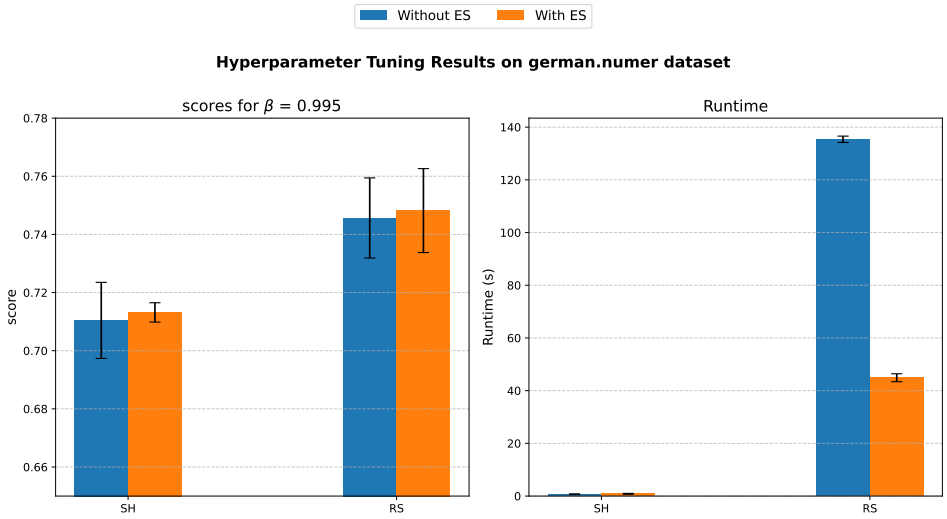


Figure 5.5: Results of running random search and successive halving with five-fold cross-validation on *german.numer* dataset using SMO with and without early stopping. The results show improved scores for both RS and SH on this dataset, highlighting the potential of ES and the impact of reducing ES frequency.

Comparing with section 4.3, where objective calculations were more frequent, we find more minor score differences for both SH and RS, partly due to reduced wall-clock time. In this setup, the gap between SH with and without ES is smaller than in section 4.3. Compared with 3 (reduced runtimes, but gains are not that significant) The score increase in this experiment mainly comes from slightly higher accuracies with ES, whereas section 4.3 showed a slight drop. While runtimes are reduced, the primary factor behind the score improvement is the better accuracies found through HPO. These results suggest that lowering ES check frequency makes early stopping less aggressive. Though overhead is lower, the impact of ES is also reduced, leading to no significant runtime speedups compared to section 4.3.

In this experiment, we observed that reducing the frequency of ES objective assessments lowers the overhead of ES in both SH and RS. We achieve comparable runtimes with and without ES for an SH procedure while still identifying well-performing models. Furthermore, the scores for both RS and SH, with and without ES, are more closely aligned than in cases where ES objective assessments occur more frequently. This highlights the benefit of applying r.

However, the reduction in runtime is not as prominent as expected. While SH benefits from the lower frequency of ES objective assessments, the runtime decreases in RS are less apparent—using r results in less aggressive early stopping. Although the reduced frequency of assessments lowers the overhead, it also decreases the likelihood of stopping early, limiting the runtime gains typically associated with ES.

The positive scores observed in this experiment stem from the less aggressive early stopping. This allows both HPO procedures to identify higher-accuracy models when using ES, leading to scores closer than those in section 4.3. This further underscores the strong bias in our setup towards maintaining accuracy over reducing runtime.

We also note that the ES configuration j_r selection does not explicitly consider the reduction in computational overhead. The scoring function evaluates only runtime reductions and accuracy, which may result in suboptimal configuration choices. Future work should investigate this trade-off further to refine the selection criteria for j_r.

5.5. Limitations and future work

A detailed analysis revealed a limitation in our experimental setup. While ES improved performance on all datasets, the Wilcoxon test did not show significance due to our experiment design. Running 100 random trials and applying Holm's multiple correction meant no results reached significance. Our analysis in section 5.3 shows that even when an ES configuration performs well on all of the used datasets, this result is not marked as significant, highlighting a weakness in our setup.

Adding more datasets to the experiment could help overcome this weakness as this increases the possible maximal value of w_{stat}, enhancing the test's statistical power. Alternatively, adjusting the experimental setup to evaluate fewer configurations would reduce the severity of Holm's correction, potentially leading to statistically significant results.

Our goal was to reduce HPO runtime while maintaining model performance. We evaluated ES configurations using a score function balancing accuracy and runtime reduction, with the β parameter controlling the trade-off. A high β prioritized accuracy, leading to more forgiving ES configurations achieving the best scores. Conversely, lower β values (emphasizing runtime reduction) favored more aggressive ES configurations. This demonstrates that our current approach leans towards a forgiving ES strategy, and the trade-off merits further exploration.

Future work should explore the balance between aggressive and forgiving ES strategies. While this study emphasizes a forgiving approach, evaluating the impact of a more aggressive ES is worthwhile. The current score function only considers runtime reduction in iterations, but alternative score functions focusing on ES overhead could offer new insights. Additionally, investigating more efficient ES objectives may enhance performance, especially given the overhead of ES calculations.

Our experimental results indicate that dimensionality does not influence the effectiveness of ES in SMO when tuning hyperparameters. However, other dataset characteristics do impact ES performance. Factors such as noise levels, class separability, and class imbalance should be studied more to better understand ES effectiveness in SMO.

6

Conclusion

This research explored the impact of incorporating early stopping (ES) into the Sequential Minimal Optimization (SMO) algorithm for hyperparameter tuning. By introducing a validation-based stopping condition, we evaluated its performance within random search (RS) and successive halving (SH) frameworks across diverse datasets, emphasizing trade-offs between runtime efficiency and model performance. This study contributes to hyperparameter optimization by demonstrating how early stopping (ES) can be strategically integrated into SMO-based SVM training to enhance efficiency while maintaining model performance.

Our results demonstrate that ES can enhance hyperparameter tuning efficiency, particularly in random search, but its effectiveness is highly dependent on dataset characteristics. We have shown that the dimensionality of the data does not impact the efficacy of early stopping. Both high-dimensional and low-dimensional datasets ($\geq$ 50 features) benefit most from aggressive early-stopping strategies. Additionally, we have shown that ES has the potential to improve RS, but in the current implementation, it does not improve SH.

The overhead from ES objective calculations outweighs its benefits only in specific cases. ES increases runtime for both SH and RS when evaluated at every iteration. This highlights a key trade-off: while early stopping shortens training, frequent objective evaluations can offset these gains, particularly in SH. To address this, we reduced the frequency of ES evaluations, successfully lowering computational overhead while still identifying well-performing models.

This study provides practical insights into the role of early stopping (ES) in hyperparameter tuning for SVMs, demonstrating that ES effectively reduces runtime in random search but offers limited gains in Successive Halving (SH) due to SH's inherent efficiency. The trade-off between accuracy and runtime, controlled by the β parameter in our scoring approach, led to forgiving ES configurations being favored. Additionally, our experimental setup, with 100 random trials and Holm's correction, limited the statistical power of the Wilcoxon test, resulting in no significant findings. Future work should address this by including more datasets and reducing the number of candidates to improve the statistical analysis.

Balancing runtime savings with computational overhead requires carefully adapted ES strategies. Future research should refine ES techniques to address the computational challenges identified here, enhancing their efficiency and applicability across diverse machine learning tasks. These insights extend beyond SVMs and could inform ES strategies for other machine learning models, offering potential benefits in real-world applications where computational efficiency is critical.

References

[1] Shawkat Ali and Kate A. Smith-Miles. “Improved support vector machine generalization using normalized input space”. In: *Lecture notes in computer science*. Jan. 2006, pp. 362–371. DOI: 10.1007/11941439_40.

[2] Luhur Bayuaji et al. “Optimization of Feature Selection in Support Vector Machines (SVM) Using Recursive Feature Elimination (RFE) and Particle Swarm Optimization (PSO) for Heart Disease Detection”. In: *2024 9th International Conference on Mechatronics Engineering (ICOM)*. 2024, pp. 304–309. DOI: 10.1109/ICOM61675.2024.10652561.

[3] Alessio Benavoli, Giorgio Corani, and Francesca Mangili. “Should We Really Use Post-Hoc Tests Based on Mean-Ranks?” In: *Journal of Machine Learning Research* 17.5 (2016), pp. 1–10.

[4] Mohamed Abdou Berbar. “Three robust features extraction approaches for facial gender classification”. In: *The Visual Computer* 30.1 (Jan. 2013), pp. 19–31. DOI: 10.1007/s00371-013-0774-8.

[5] Edward Bergman, Lennart Purucker, and Frank Hutter. *Don’t Waste Your Time: Early Stopping Cross-Validation*. en. May 2024.

[6] James Bergstra and Yoshua Bengio. “Random Search for Hyper-Parameter Optimization”. In: *Journal of Machine Learning Research* 13.10 (2012), pp. 281–305.

[7] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006. ISBN: 978-0387-31073-2.

[8] Léon Bottou and Olivier Bousquet. “The Tradeoffs of Large Scale Learning”. In: *Advances in Neural Information Processing Systems*. Ed. by J. Platt et al. Vol. 20. Curran Associates, Inc., 2007.

[9] Léon Bottou and Chih-Jen Lin. “Support Vector Machine Solvers”. In: *Large Scale Kernel Machines*. Ed. by Léon Bottou et al. Cambridge, MA.: MIT Press, 2007, pp. 301–320.

[10] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[11] Jair Cervantes et al. “A comprehensive survey on support vector machine classification: Applications, challenges and trends”. In: *Neurocomputing* 408 (2020), pp. 189–215. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2019.10.118.

[12] Chih-Chung Chang and Chih-Jen Lin. “LIBSVM”. In: *ACM Transactions on Intelligent Systems and Technology* 2.3 (Apr. 2011), pp. 1–27. DOI: 10.1145/1961189.1961199.

[13] Olivier Chapelle. “Training a support vector machine in the primal”. In: *The MIT Press eBooks*. Aug. 2007, pp. 29–50. DOI: 10.7551/mitpress/7496.003.0004.

[14] Janez Demšar. “Statistical Comparisons of Classifiers over Multiple Data Sets”. In: *Journal of Machine Learning Research* 7.1 (2006), pp. 1–30.

[15] Radwa Elshawi, Mohamed Maher, and Sherif Sakr. *Automated Machine Learning: State-of-The-Art and Open Challenges*. en. June 2019.

[16] Rong-En Fan, Pai-Hsuen Chen, and Chih-Jen Lin. “Working Set Selection Using Second Order Information for Training Support Vector Machines”. In: *Journal of Machine Learning Research* 6.63 (2005), pp. 1889–1918.

[17] Rong-En Fan et al. *LIBLINEAR: a library for large linear classification*. Ed. by Soeren Sonnenburg. Vol. 9. Aug. 2008, pp. 1871–1874.

[18] Adrian Cătălin Florea and Răzvan Andonie. “A dynamic early stopping criterion for random search in SVM hyperparameter optimization”. In: *IFIP advances in information and communication technology*. Jan. 2018, pp. 168–180. DOI: 10.1007/978-3-319-92007-8_15.

[19] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[20] Rosita Guido, Maria Carmela Groccia, and Domenico Conforti. “A hyper-parameter tuning approach for cost-sensitive support vector machine classifiers”. In: *Soft Computing* 27.18 (Feb. 2022), pp. 12863–12881. DOI: 10.1007/s00500-022-06768-8.

[21] Rosita Guido et al. “An Overview on the advancements of support vector machine models in healthcare Applications: a review”. In: *Information* 15.4 (Apr. 2024), p. 235. DOI: 10.3390/info15040235.

[22] Nyoman Gunantara. “A review of multi-objective optimization: Methods and its applications”. In: *Cogent Engineering* 5.1 (Jan. 2018), p. 1502242. DOI: 10.1080/23311916.2018.1502242.

[23] Daniel Horn et al. *A comparative study on large scale kernelized support vector machines - advances in data analysis and classification*. July 2016.

[24] Chen-Chiung Hsieh and Dung-Hua Liou. “Novel Haar features for real-time hand gesture recognition using SVM”. In: *Journal of Real-Time Image Processing* 10.2 (Nov. 2012), pp. 357–370. DOI: 10.1007/s11554-012-0295-0.

[25] Chih-Wei Hsu and Chih-Jen Lin. *A simple decomposition method for support vector machines - machine learning*. 2002.

[26] Bootan M. Hussein and Shareef M. Shareef. “An Empirical Study on the Correlation between Early Stopping Patience and Epochs in Deep Learning”. In: *ITM Web of Conferences* 64 (Jan. 2024), p. 01003. DOI: 10.1051/itmconf/20246401003.

[27] Kevin Jamieson and Ameet Talwalkar. *Non-stochastic best arm identification and hyperparameter optimization*. en. Feb. 2015.

[28] S. S. Keerthi et al. “Improvements to Platt’s SMO Algorithm for SVM Classifier Design”. In: *Neural Computation* 13.3 (Mar. 2001), pp. 637–649. DOI: 10.1162/089976601300014493.

[29] Aaron Klein et al. *Fast Bayesian optimization of machine learning hyperparameters on large datasets*. en. Mar. 2016.

[30] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. “Numba: a LLVM-based Python JIT compiler”. In: LLVM ’15. Austin, Texas: Association for Computing Machinery, 2015. ISBN: 9781450340052. DOI: 10.1145/2833157.2833162.

[31] Lisha Li et al. *Hyperband: A novel Bandit-Based approach to hyperparameter optimization*. en. Mar. 2016.

[32] Zachary C. Lipton. *The mythos of model interpretability*. en. June 2016.

[33] Jason L. Loeppky, Jerome Sacks, and William J. Welch. “Choosing the sample size of a computer experiment: A practical guide”. In: *Technometrics* 51.4 (Nov. 2009), pp. 366–376. DOI: 10.1198/tech.2009.08040.

[34] Rafael G. Mantovani et al. “Effectiveness of Random Search in SVM hyper-parameter tuning”. In: *2015 International Joint Conference on Neural Networks (IJCNN)*. 2015, pp. 1–8. DOI: 10.1109/IJCNN.2015.7280664.

[35] Shili Peng et al. “Regression-Based Hyperparameter Learning for Support Vector Machines”. In: *IEEE Transactions on Neural Networks and Learning Systems* 35.12 (2024), pp. 18799–18813. DOI: 10.1109/TNNLS.2023.3321685.

[36] Matthew Penn and Chris Milroy. “Powering Practical Performance: Accelerated Numerical Computing in Pure Python”. In: *2022 IEEE High Performance Extreme Computing Conference (HPEC)*. 2022, pp. 1–5. DOI: 10.1109/HPEC55821.2022.9926309.

[37] John Platt. *Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines*. Tech. rep. MSR-TR-98-14. Microsoft, Apr. 1998.

[38] Lutz Prechelt. “Early stopping — but when?” In: *Lecture notes in computer science*. Jan. 2012, pp. 53–67. DOI: 10.1007/978-3-642-35289-8_5.

[39] Sebastian Salazar, Samuel Denton, and Ansaf Salleb-Aouissi. *Counterfactual explanations for support vector machine models*. en. Dec. 2022.

[40] Shai Shalev-Shwartz et al. *Pegasos: Primal estimated sub-gradient solver for SVM - Mathematical Programming*. Oct. 2010.

[41] Vivian S. Silva, André Freitas, and Siegfried Handschuh. *On the Semantic Interpretability of Artificial Intelligence Models*. en. July 2019.

[42] Ryuta Tamura, Yuichi Takano, and Ryuhei Miyashiro. *Feature subset selection for kernel SVM classification via mixed-integer optimization*. en. May 2022.

[43] Cheng-Hao Tsai, Chieh-Yen Lin, and Chih-Jen Lin. "Incremental and decremental training for linear classification". In: *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '14. New York, New York, USA: Association for Computing Machinery, 2014, pp. 343–352. ISBN: 9781450329569. DOI: 10.1145/2623330.2623661.

[44] Chih-Chung Chang National Taiwan University et al. *LIBSVM: A library for support vector machines: ACM Transactions on Intelligent Systems and Technology: Vol 2, no 3*. Apr. 2011.

[45] Vladimir Vapnik. *Estimation of dependences based on empirical data*. 2006.

[46] Siqi Wang et al. "Hyperparameter selection of one-class support vector machine by self-adaptive data shifting". In: *Pattern Recognition* 74 (2018), pp. 198–211. ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2017.09.012.

[47] Zeyi Wen et al. "ThunderSVM: A Fast SVM Library on GPUs and CPUs". In: *Journal of Machine Learning Research* 19.21 (2018), pp. 1–5.

[48] Jason Weston et al. "Feature Selection for SVMs". In: *Advances in Neural Information Processing Systems*. Ed. by T. Leen, T. Dietterich, and V. Tresp. Vol. 13. MIT Press, 2000.

[49] Stephen J. Wright. *Coordinate Descent Algorithms*. en. Feb. 2015.

[50] Xiao Zhang, Mohammad H. Mahoor, and S. Mohammad Mavadati. "Facial expression recognition using l_p-norm MKL multiclass-SVM". In: *Machine Vision and Applications* 26.4 (Apr. 2015), pp. 467–483. DOI: 10.1007/s00138-015-0677-y.

A

SMO Profiling

This appendix is a supplement to section 3.1, where we give the implementation details of our version of the Sequential Minimal Optimization algorithm (SMO). First, we compare the runtimes of the three versions of the SMO algorithm we implemented in Python.

We use the *w1a* dataset in section 4.3 for all the experiments. We perform every experiment using 5-fold stratified cross-validation similar to the other experiments, where we also split the data according to the setup proposed in section 3.2. In this experiment, we won't focus on hyperparameter tuning but instead on the performance of the SMO algorithm for individual models. Thus, we perform a random search using 100 candidates to find hyperparameters for an RBF kernel SVM that works well. As hyperparameter ranges, we use the same bounds for C, ϵ and γ as used in section 3.2.

A.1. SMO version comparison

Multiple versions of the SMO algorithm have been proposed, and we implemented several to study their behaviour. In this experiment, we compare the performance of our implementation of Keerthi's SMO version (as proposed in [28]) with the implementation from [16], as well as our optimized version of the latter algorithm.

Section A.1.1 describes the different SMO versions we implemented, while Section A.1.2 presents the experimental results comparing the performance of these implementations.

A.1.1. Methodology

This experiment compares the runtime of different implementations of the Sequential Minimal Optimization (SMO) algorithm for training a model. Using a costly heuristic, Platt's original SMO algorithm [37] selects the working set (two alphas to optimize). Keerthi et al. [28] improve this by introducing the maximal violating pair approach, which maximizes the objective increase. The LibSVM implementation [44], based on Fan et al. [16], further refines working set selection using second-order information, leading to faster convergence.

We include LibSVM's SMO implementation in our comparison to evaluate its performance against our implementations. While LibSVM follows the approach of [16], it incorporates additional optimizations, such as shrinking. Moreover, LibSVM is implemented in C++, whereas our version runs in Python. Since C++ is natively faster, we also consider the impact of language-specific optimizations [36].

We implemented Keerthi's SMO algorithm and two versions of Fan et al.'s algorithm. The first follows the pseudocode from [16] without modifications. The second incorporates efficiency improvements to reduce runtime while preserving convergence behaviour. These optimizations focus solely on computational efficiency, not altering SMO's effectiveness.

One key optimization is precomputing the kernel and Q-matrices. While storing both in memory can be challenging for large datasets, it enables batch kernel evaluations and efficient matrix multiplications in Python, significantly accelerating the algorithm.

We vectorized the gradient update step, leveraging the precomputed kernel matrix to update the entire Gradient in a single operation, further improving efficiency.

Python's interpreter overhead can limit scalability on large datasets. To address this, we use Numba [30] to compile critical functions into machine code, achieving performance comparable to compiled languages. Optimized functions include RBF kernel computation, model prediction, and working set selection—operations that are frequently called and benefit significantly from Numba's speedup.

We train four models using different versions of the SMO algorithms. The selected hyperparameters for the RBF kernel SVMs are $C = 2796.28$, $\epsilon = 0.004713$, and $\gamma = 0.0001076$. Training runs without iteration limits or early stopping. For comparison, we measure the number of iterations required for convergence and the total wall-clock time. The final accuracy of each model is tested on the test data set.

A.1.2. Results

Figure A.1 presents the experimental results, showing that Fan et al.'s SMO implementation achieves faster convergence in iterations and wall-clock time. This is expected, as the key improvement in [16] involves selecting two alphas that yield better objective improvements, reducing the number of iterations required.

Fewer iterations also translate to lower wall-clock time, as shown in the right plot of Figure A.1. This plot further highlights how our optimizations significantly reduce runtime. While requiring the same number of iterations, our optimized SMO version runs considerably faster than the unoptimized version.

Notably, this experiment teaches us that LibSVM further optimizes the SMO algorithm. Figure A.1.2 shows how LibSVM uses significantly fewer iterations than Fan's version of the SMO algorithm. This is interesting because the library does implement Fan's SMO algorithm [12]. We also note how LibSVM completes the fastest of the four tested SMO versions in this experiment.

The different implementations yield varying final accuracies. Keerthi's algorithm produces models with an accuracy of $93.82 \pm 1.46\%$, while Fan's version achieves higher performance with fewer iterations, scoring $97.13 \pm 0.34\%$. Notably, our optimized version matches the accuracy of the unoptimized Fan et al. version, indicating that our improvements did not affect the SMO algorithm's convergence behaviour. Additionally, we see that Fan's algorithm leads to a slightly higher accuracy than LibSVM, where LibSVM has an accuracy of $97.05 \pm 0.7\%$.

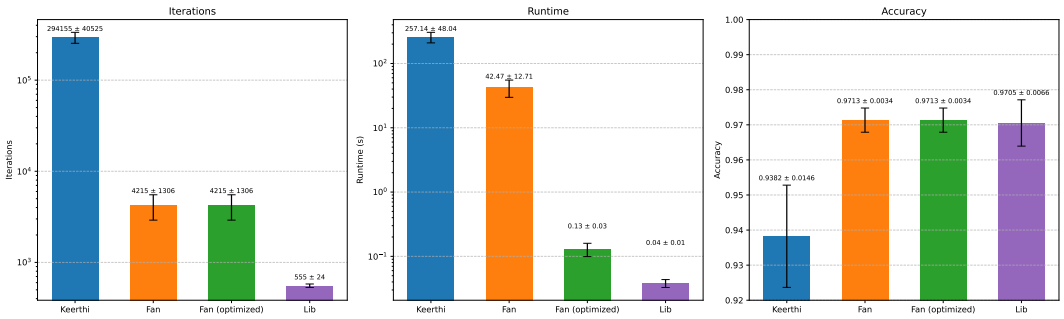


Figure A.1: Performance comparison of four SMO implementations on the w1a dataset. The plot presents the number of iterations, runtime (in seconds), and model accuracy. Our optimized version reduces runtime while maintaining the same number as Fan's implementation. LibSVM remains the fastest solver, converging in fewer iterations than both Fan's and Keerthi's implementations.

This experiment reveals performance differences among SMO implementations. Fan et al.'s version converges fastest in both iterations and wall-clock time. The optimized variant enhances efficiency without modifying the SMO algorithm.

A key limitation is using a single dataset, as SMO's behaviour is highly dataset-dependent. Additionally, hyperparameters significantly affect runtime. Thus, differences in runtime and iterations may vary under different experimental setups.

B

Hinge Loss as an Early Stopping Metric

This appendix explores hinge loss as an early stopping metric in the experiment outlined in Section 3.2. We define the objective function employed for early stopping in Section B.1. Subsequently, Section B.2 presents the results obtained from the experiment.

B.1. Hinge loss objective

We aim to compute the total hinge loss over the validation set. The rationale is that the SMO algorithm inherently minimizes hinge loss as it forms the foundation of the model's optimization process. When a reduction in hinge loss is no longer observed, we assume that training can be halted. Let $f(x_i)$ denote the model's i-th sample prediction. Using the hinge loss definition from Equation (2.3), we define the following objective function:

$$O_{val} = \frac{1}{n_{val}} \sum_{i=1}^{n_{val}} \max\{0, 1 - t_i f(x_i)\} \tag{B.1}$$

Using the adjusted objective function, we express the remaining patience p_{cur} as follows:

$$p_{cur} = \begin{cases} p_{cur} - 1 & \text{if } O_{val}^{l} - O_{val}^{best} \leq -\epsilon_{es} \\ p & \text{if } O_{val}^{l} - O_{val}^{best} > -\epsilon_{es} \end{cases} \tag{B.2}$$

Note how, in this case, we use the negative value of the early-stopping tolerance ϵ_{es}. This is necessary since the hinge loss is a decreasing objective compared to the accuracy metric.

B.2. Results and Discussion

Figures B.1 and B.2 show the results of applying early stopping in a random search training SVMs with the SMO algorithm. Similar to using the accuracy objective, we do, in this case, also see early stopping configurations that reduce the runtime of a hyperparameter tuning procedure while still finding good-performing models. This behaviour occurs for both linear and RBF kernel SVMs.

In this setting, the Wilcoxon test with Holm's multiple testing correction fails to detect configurations that work well across all six datasets. Again, this indicates that the effectiveness of an early-stopping configuration strongly depends on the dataset in which it is used.

However, when we observe figures B.3 and B.4, we find configurations that perform well across all six datasets. This is particularly interesting because this is not the case when using accuracy as the objective for early stopping. Additionally, using the Hinge loss does result in configurations that work well for the *australian* dataset when using RBF kernels, as shown in figure B.4. This indicates that using hinge loss as early-stopping has advantages over using the accuracy metric.

Figures B.3 and B.4 show the five configurations that have, on average, the best score over the six datasets. Different from the results in section 4.1, we see configurations with a positive score for all datasets. These results indicate that the hinge loss on the validation data might better demonstrate the training progress.

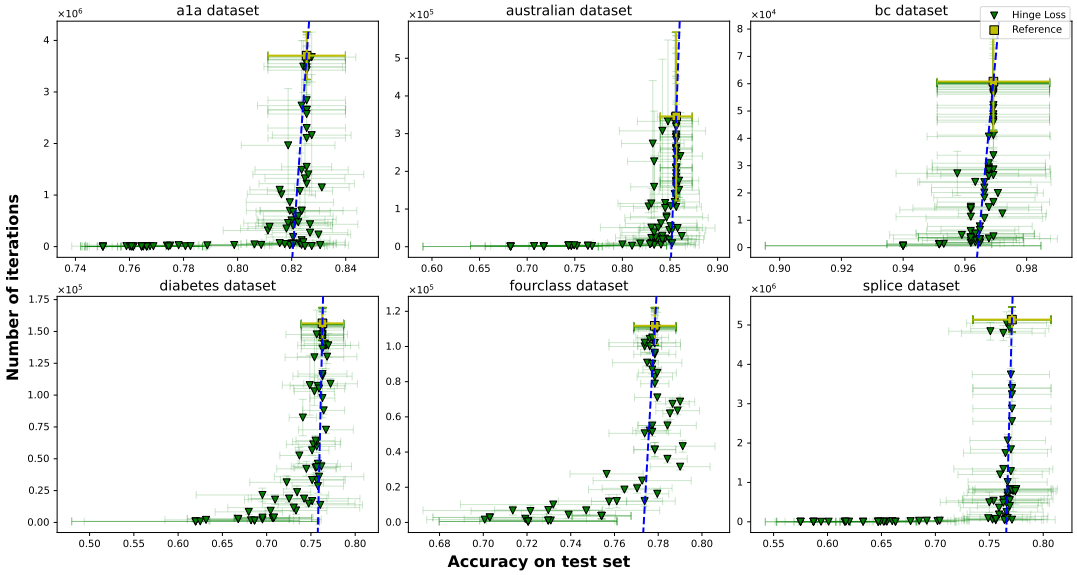


Figure B.1: Pareto front plots displaying the total iterations of the full hyperparameter tuning procedure for linear SVMs, with the X-axis representing the accuracy of the best-found model. The dashed blue line marks the score threshold; any S_j^i to the right of this line indicates an improved score compared to S_{ref}^i. The results demonstrate that ES can enhance performance for all six datasets.

Additionally, figure B.4 shows that all found configurations perform well on the *australian* dataset. When using accuracy as an objective, this is not the case, showing us that there are

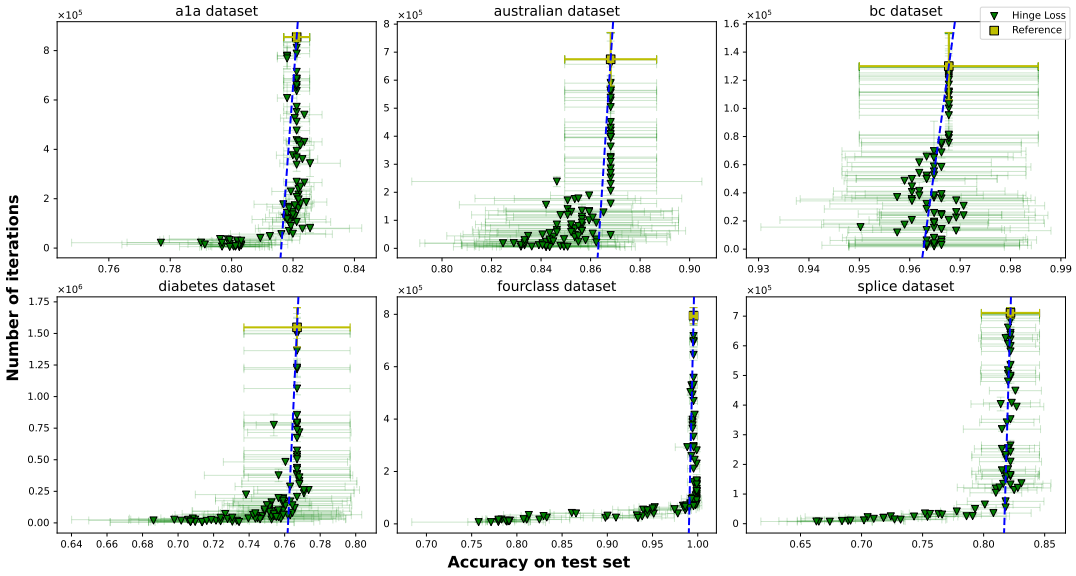


Figure B.2: Pareto front plots displaying the total iterations of the full hyperparameter tuning procedure for RBF SVMs, with the X-axis representing the accuracy of the best-found model. The dashed blue line marks the score threshold; any S_j^i to the right of this line indicates an improved score compared to S_{ref}^i. The results demonstrate that ES can enhance performance for all six datasets.

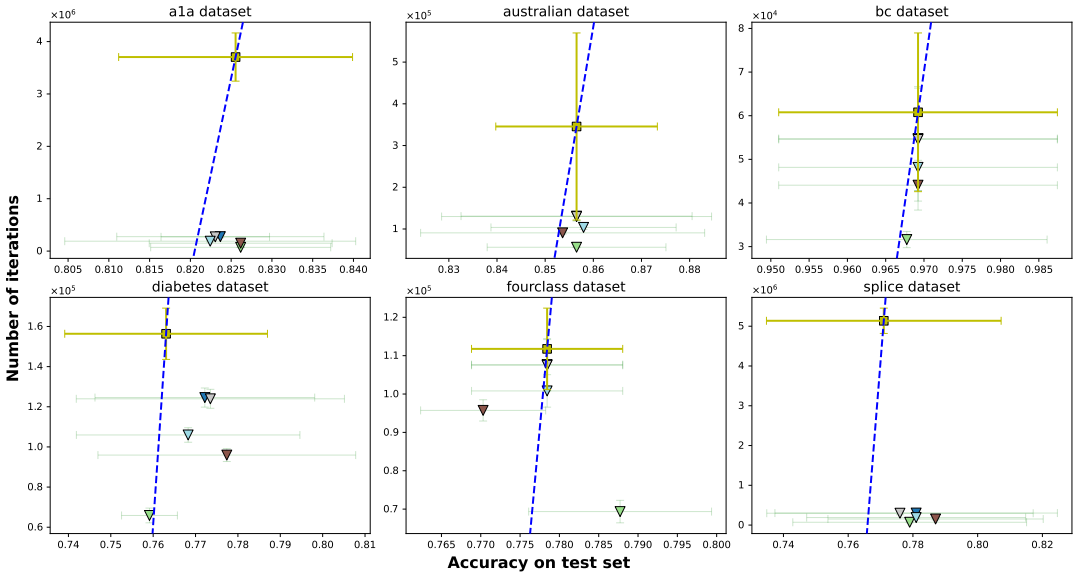


Figure B.3: Pareto front plot showing S_{ref}^i compared to the five S_j with the highest average score over the six datasets using the accuracy as an early stopping metric and linear kernels. The plot also shows the score line for $\beta = 0.995$.

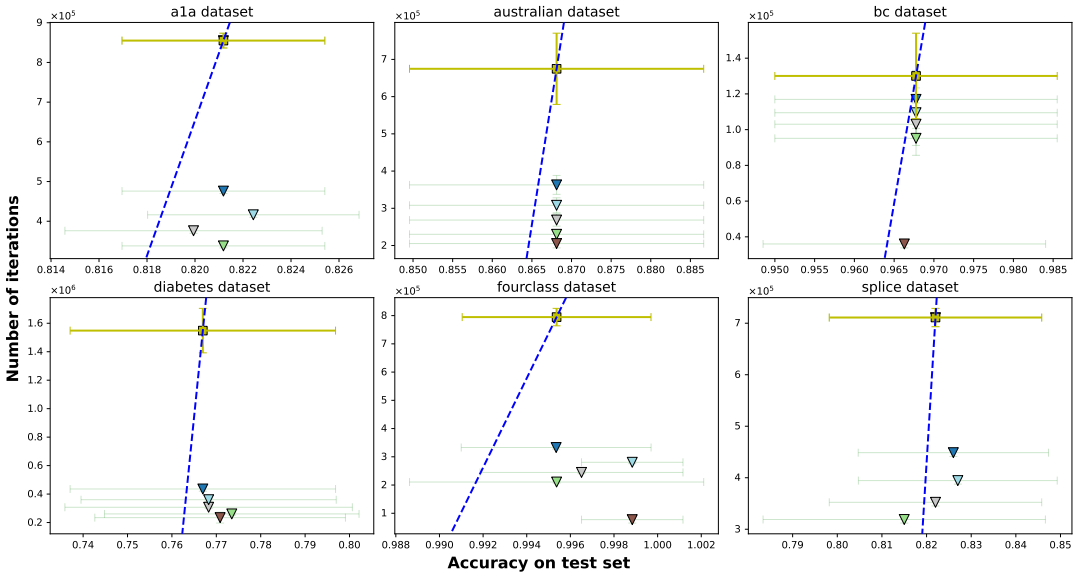


Figure B.4: Pareto front plot showing S^i_{ref} compared to the five S_j with the highest average score over the six datasets using the accuracy as an early stopping metric and RBF kernels. The plot also shows the score line for $\beta = 0.995$.