Evolution of a Support Vector Machine

Sebastian Baron
Faculty of Natural Science
Paris Lodron Universität Salzburg
Salzburg, Austria
sebastian.baron@stud.sbg.ac.at

Julia Himmelsbach
Faculty of Natural Science
Paris Lodron Universität Salzburg
Salzburg, Austria
julia.himmelsbac@stud.sbg.ac.at

Simon Valverde
Faculty of Natural Science
Paris Lodron Universität Salzburg
Salzburg, Austria
simon.valverde@stud.sbg.ac.at

Matthias Wintersteller
Faculty of Natural Science
Paris Lodron Universität Salzburg
Salzburg, Austria
matthias.wintersteller@stud.sbg.ac.at

ABSTRACT

In this paper we propose a support vector machine (SVM) constructed with the help of an evolutionary algorithm to classify various data sets (e.g., ionosphere, handwritten digits and the wall following robot data).

SVMs itself are supervised learning models which have shown good results classifying huge amount of high-dimensional data by constructing a hyper-plane that divides training objects into classes.

To optimize the hyper-parameters of the SVM an evolutionary algorithm (EA) was chosen which basically follows the principles of biological evolution. Hence, the EA uses mechanisms such as reproduction, mutation and recombination to develop better parameters for the SVM.

1 INTRODUCTION

As the use of predictive algorithms gains more and more practical relevance, it makes sense to make the predictions of these algorithms as precise as possible. A common problem in this context is the optimal setting of hyper-parameters which may differ for each problem. Although default values in most libraries already provide a good classification, this can be further improved by proper hyper-parameter tuning.

Hyper-parameter optimization or hyper-parameter tuning aims to automatically choose a set of optimal hyper-parameters for a learning algorithm. A hyper-parameter is a parameter whose value is used to control the learning process.

This has a high application demand in practice, as it allows efficiency to be increased. One method to optimize the hyper-parameters are meta-heuristics like evolutionary algorithms (EAs). This paper proposes a support vector machine (SVM) combined with an evolutionary algorithm to optimize the hyper-parameters of the SVM. The evolutionary algorithm enables the optimization of the hyper-parameters.

In addition, it is also capable of learning with different kernels. We demonstrate on benchmark data sets that the EA improves the performance achieved by the SVM.

Section 2 and 3 give a brief introduction about support vector machines and evolutionary algorithms. After this, section 4 presents the used data sets, the pre-processing methods and the implementation. Finally, section 6 presents the results.

2 SUPPORT VECTOR MACHINE

The main idea of a SVM is to map the input vector to a feature space, different from the input data, induced by a kernel and to classify the transformed data by a linear function [5]. Consequently, the objective of a support vector machine is to find a hyperplane which divides training objects into classes. Furthermore, the hyperplane should have maximum distance to the nearest training patterns of any class (support vectors) [12]. This is because, in general, the larger the margin, the smaller the generalization error of the classifier. The wide, empty border ensures that objects that do not correspond exactly to the training objects are classified as reliably as possible [12].

Additionally, the SVM supports linear classification as well as nonlinear classification. Linear classification results in the hyperplane being placed at the greatest possible distance between to samples. In nonlinear classification, the problem is transformed into a linear classification problem in a high-dimensional space. For the nonlinear classification the kernel-method is used which operates in a higher dimension. This is achieved by transforming the data into a higher-dimensional space because of that more complex hyper planes can be realized with which one hopes for better separability [12]. The choice of the optimal kernel function is considered as "crucial step in handling a learning task with an SVM" [5], because it influences the training accuracy and the generalization performance as well as the run time and storage complexity during and after training [6].

In an easy form SVMs are applied on binary classification dividing the data points either in 1 or 0. For mulitclass classification the problem is broken down into multiple binary classification cases, which is also called one-vs-one approach. Furthermore each classifier separates points of two different classes and comprising all one-vs-one classifiers leads to a multiclass classifier. To classify m classes of a data set, one-vs-one uses $\frac{m(m-1)}{2}$ SVMs [2]. Besides one-vs-one, there exists the one-to-rest approach. In that approach, the breakdown is set to a binary classifier per each class. This approach needs m SVMs for m classes in a data set [2].

Another point that needs to be considered when dealing with SVMs is the C parameter. C, the regularization parameter, controls the trade off between empirical error and complexity of the hypothesis space used. A default value for C is often 1 but if there are many outliers a lower C would be better so the SVM can generalize in a better way.

Likewise, another parameter is the stopping critera which tells the algorithm when to stop searching for a minimum (or maximum) once some tolerance is achieved.

3 EVOLUTIONARY ALGORITHMS

Evolutionary algorithms (EA) are meta-heuristics for optimization problems based on the principles of biological evolution. Meta-heuristics return approximate solutions in contrast to analytical procedures. The basic principle of biological evolution can be formulated like this [8]:

> *Advantageous characteristics resulting from random variation are favored by natural selection.*

In other words, individuals with advantageous characteristics have better chances to create offspring and multiply these traits [8].

Within an EA the solution candidates are encoded as chromosomes. The typical procedure follows the following steps:

(1) Generation of random solutions
(2) Evaluation of these solutions
(3) Recombination of good solutions and mutations
(4) Repeat until termination criterion is met

The main principles used in EAs are fitness, selection, mutation and crossover which are described in the next chapters.

3.1 Fitness

The fitness function evaluates how good a solution candidate is compared to the other solution candidates, in other words it determines how fit a solution candidate is.

For that reason, we have to evaluate the fitness of each chromosome, in our case the the set of hyper-parameters, for the SVM.

3.2 Selection

The selection method determines which individuals are selected to produce the descendants of the population. Examples are the expected value model, elitism, roulette wheel selection, rank selection and tournament selection. The roulette wheel selection is a fitness proportionate selection method due to the selection with a probability according to the specific fitness value of an individual. Another method is the tournament selection in which two or more individuals compete against each other in a tournament. The individual with the highest fitness value wins the tournament and is selected into the next generation. In this method, the selection pressure can be controlled by adjusting the tournament size [8].

3.3 Crossover

Genetic operators that operate on two parent individuals are commonly called crossover operators. The crossover operator is analogous to reproduction and biological crossover. In crossover more than one parent is selected and one or more off-springs are produced using the genetic material of the parents. One example is the so called one-point crossover. In this procedure, a random point is first selected. The next step is to cut both parent chromosomes at this point and then to combine the first part of one parent with the second part of the other parent. In this process, two new chromosomes are created that are different from the parents. Crossover enables the EA to improve the fitness generation per generation. Further examples for recombination methods are N-point crossover, uniform crossover, shuffle crossover and diagonal crossover [8].

3.4 Mutation

Besides selection, mutation is another very important principle in evolution. In simple terms, mutation may be defined as a small random change in the chromosome, in order to get a new solution. It is used to maintain and introduce diversity in the genetic population and is usually applied with a low probability. A simple example for mutation is the standard mutation which replaces a current allele of a gene by another. An allele is a "value" of a gene. It is called bit mutation if the allele is binary and the randomly chosen allele gets inverted. For chromosomes that are vectors of real numbers one can use Gaussian mutation. It adds to each gene a random number drawn from a normal or Gaussian distribution. To achieve this, each chromosome is equipped with either one variance for all genes or a vector of variances each corresponding to one gene. Further methods, are for example pair exchange, shuffling/permuting a part of a chromosome, and reversing a part of a chromosome [8].

3.5 Local variance adaptation

A problem in evolutionary algorithms might be that chromosomes with undesirable variances create too many undesired descendants. A variance is undesirable if it is either too small and as a result the chromosomes do not develop fast enough or the variance is too large and as a result the chromosomes move too far away from their parents. Too much or too little variance could result in too early extinction.

Therefore, local variance adaption can be used. It adds a variance to the chromosomes. Either, one variance for all vector elements or an individual variance for each vector element. Individuals with undesired variances will produce descendants with lower fitness. As a consequence their genes will die out.

3.6 Termination criterion

Furthermore, the EA needs a termination criterion. Basically there are three termination conditions that are used:

(1) when an upper limit on the number of the generations is reached or (2) when an upper limit on the number of the evaluations of the fitness function is reached or (3) when the chance of achieving a significant change is very low (no improvement was observed for a number of generations).

4 IMPLEMENTATION

4.1 Software

The programming language which was used for that project was python with the common used machine learning package scikit-learn, especially the class *sklearn.svm.SVC*. We developed an object-oriented implementation with two classes: population and solution candidates.

4.2 Data

Since the goal of this work is to investigate the effects of hyper parameter optimization, it is significant to consider multiple data sets. Therefore, we use three open data sets. This section gives a brief overview over the used data sets wall-following robot, ionosphere and handwritten digits. The data sets can be downloaded from the UC Irvine Machine Learning Repository [4].

4.2.1 Robot. The data was collected by sensors of a robot navigate through space by following a wall in clockwise direction. The robot has 24 ultrasonic sensors arranged in a circle around its "waist". The data set consists of 5456 instances. The data has the classifications: [4]

- Move-Forward (2205 samples or 40.41%)
- Slight-Right-Turn (826 samples or 15.13%)
- Sharp-Right-Turn (2097 samples or 38.43%)
- Slight-Left-Turn (328 samples or 6.01%)

each corresponding to the desired action in a given state. The SVM attempts to infer correct robot behavior from this sensor data.

4.2.2 Ionosphere. This data set contains radar measurements of 351 instances. It has 34 real value features and a class attribute which is either "good" ("g") or "bad" ("b"). Target variable are the free electrons in the ionosphere. "Bad" radar signals pass through the ionosphere, "good" does not. The "good" signals show evidence of some type of structure in the ionosphere. 225 instances are labeled as "good" and 126 as "bad".

4.2.3 Handwritten digits. The handwritten digits data set contains 10 classes where each class refers to a digit from 0 to 9. Furthermore, there are 64 attributes which are based on an 8x8 image matrix where each element is an integer in the range of 0 up to 16. Additionally, the number of instances is 5620 [4].

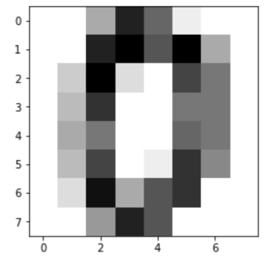
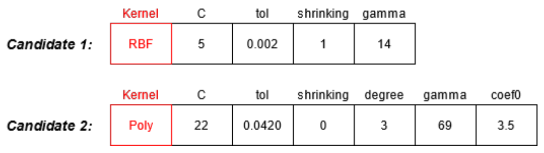


Figure 1: Example of a digit ("0") from the handwritten digits data set

4.3 Encoding and Initialization

Generally speaking, each individual represents a solution candidate for the optimization problem, meaning that it contains a set of hyper-parameters for a SVM. One hyper-parameter that needs special handling is the kernel function since different kernel functions require different hyper-parameters. e.g. the degree of the polynomial kernel function is only available for the polynomial kernel function (see table 1). An illustration of the candidate encoding is shown below.



	Kernel	C	tol	shrinking	gamma
Candidate 1:	RBF	5	0.002	1	14

	Kernel	C	tol	shrinking	degree	gamma	coef0
Candidate 2:	Poly	22	0.0420	0	3	69	3.5

Figure 2: Illustration of candidate encoding

As a consequence, the kernel function of a candidate is determined upon initialization and not changed in the further process, while all other hyper-parameters are initialized randomly and affected by the genetic operations and therefore subject to the evolution process. All the parameters that are considered are listed in table 2) An important parameter is C. It controls the trade-off between margin maximization and error minimization [3].

Attribute	Type	Description
C	float	Regularization parameter
kernel	string	Specifies the kernel type
degree	int	Degree of polynomial kernel function
gamma	float	Kernel coefficient
coef0	float	Independent term in kernel function
shrinking	bool	Whether to use shrinking heuristic
tol	float	Tolerance for stopping criterion.

Table 1: Attribute description of *sklearn.svm.SVC* [11]

As mentioned before the choice of the kernel function is acknowledged as very important. In practice, only a few different types of kernels have been used. The reason are difficulties due to tuning the parameters [3]. The kernel functions we used were:

(1) Linear function
(2) Radial basis function (RBF)
(3) Polynomial function
(4) Sigmoid function

Since the kernel function is not subject to genetic operations, it is important to ensure a fair distribution of kernel functions during the initialization so that the evolution can decide which kernel is favorable (and thus will create more offspring) and which is not (those chromosomes are expected to die out). As our experiments have shown, fair distribution dies not mean that each kernel function should be represented in the population with equal proportion

Therefore the initial population consists of 6 % linear functions, 12 % radial basis functions, 62 % polynomial functions and 20 % sigmoid functions. The different number of functions is due to the fact that linear functions run into the local optimum faster. In

Attribute	Linear	RBF	Poly	Sigmoid
C	✓	✓	✓	✓
tol	✓	✓	✓	✓
shrinking	✓	✓	✓	✓
degree	×	×	✓	×
gamma	×	✓	✓	✓
coef0	×	×	✓	✓

Table 2: Attributes and kernel functions [11]

order that the others do not die off too quickly, we optimized the distribution for the ionosphere data set.

4.4 Fitness

The evaluation is performed after the individual SVMs with their candidate parameters have been calculated on each data set. As re-sampling method, we used cross-validation with three randomly divided data sets. The data set is split randomly into training and test data for three times. For each of the three we use the training data to train the SVM and the remaining test data to evaluate the SVM. The mean accuracy of all three groups is equal to the fitness of an individual. Accuracy is defined as

$$Accuracy = \frac{TP + TN}{(TP + FP + TN + FN)}$$

where TP, TN, FP and FN are the number of correctly predicted positives, correctly predicted negatives, wrongly predicted positives and wrongly predicted negatives. In case of multi-label classification, the SVM uses an one-vs-one approach as explained in chapter 2 [11]. To classify m classes, $\frac{m(m-1)}{2}$ classifiers are constructed. Each classifier trains data from two classes.

4.5 Selection

For the selection the algorithm implemented μ, λ and $\mu + \lambda$ *Selection*. In each iteration only the best individuals (as measured by their fitness score) are passed on to the next generation. In general, there are two possible strategies:

- **Plus strategy**:
  The individuals of the next generation are selected from the pool of parent generation and their offspring (generated by mutation and crossover).
- **Comma strategy**:
  The individuals of the next generation are strictly selected from the offspring of the parent generation.

The plus strategy usually leads to better results in our experiments. In general, the comma selection performs better than plus selection (for the same μ and λ) [1] [7][10]. The reason is that on one hand the comma strategy fails to recall the parent values after each generation which enables a temporary degradation of the fitness and on the other hand that the plus strategy is more likely to be caught in a local optimum, especially when the population is small [10]. According to [7] the comma strategy is inefficient on every function with a unique optimum for smaller offspring populations. For a larger λ both strategies have similar results [7]. A reason why our experiments showed better results for the plus strategy could be that we had only a small offspring population.

4.6 Crossover

The implementation uses uniform crossover. As the activation functions have different attributes, the population has to be filtered according to their type of kernel function. Then two candidate are randomly selected from the filtered list and combined. Afterwards remove these candidates from the list. Figure 3 visualizes uniform crossover.

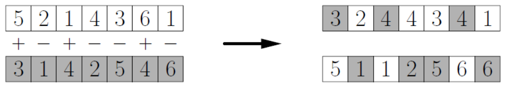


Figure 3: Uniform Crossover after [8].

4.7 Mutation

The implemented method first adjusts the local variance. See more about this in the next chapter (4.8). For each parameter the method randomly alters one or more genes in a chromosome. As most parameters are real-valued numbers we use gaussian mutation. Only the parameter "shrinking" is a boolean. Therefore, we use bit mutation.

4.8 Local variance adaptation

We use local variance adjustment because we are not in a discrete space, i.e. the options are "infinite". Each individual is mutated with a certain probability. The mutation consists of addition of a random vector with $x_i \sim \mathcal{N}(0, \sigma_i)$. σ_i is an individual variance which is specific for the respective gene (local). This local variance is changed in each generation as follows:

$$\sigma_i^{(\mathrm{new})} = \sigma_i^{(\mathrm{old})} \cdot \exp(r_1 \cdot N(0,1) + r_2 \cdot N_i(0,1))$$

Where $\mathcal{N}(0,1)$ is a normally distributed random number per chromosome and $\mathcal{N}_i(0,1)$ is a normally distributed random number for each vector element. Recommended values for the parameter r_1 and r_2 are $r_1 = 0.1$ and $r_2 = 0.2$ [9].

The expectation is that individuals with undesired variance will produce offspring with lower fitness values and their genes will die out as a consequence.

4.9 Termination Criterion

The implementation has two termination criteria: a maximum number of generations and a maximal fitness value. If one is reached, the algorithm is stopped.

5 EXPERIMENTAL SETUP

To enable good reproducibility, we have refrained from feature engineering and filtering. In summary, the overall setup of the evolutionary algorithm was as follows: We initialized a population of size 100, with a maximal number of 25 generations to be computed. The parameters to be optimized are listed in table 2. The probabilities that an individual was subjected to mutation or crossover was 15% respectively and the individuals of the next generation was determined using the +-strategy. As stated before, once the first individual reached an accuracy value of 100% on the training data the algorithm was terminated.

6 RESULTS

For the evaluation we used the data sets wall-following robot, ionosphere and handwritten digits. To be able to compare the results, we constructed a SVM for each of the data sets with the standard parameters of the class *sklearn.svm.SVC* and with the EA. Table 3 gives an overview above all achieved accuracy's.

Accuracy	[optimized parameters]	[standard parameters]
Ionosphere	0.952	0.906
Handwritten digits	0.992	0.925
Wall-following robot	0.952	0.906

Table 3: Accuracy's with the evolutionary optimized hyper-parameters and with the standard parameters

In all runs the algorithm used 25 generations and 100 individuals per population. With the ionosphere data set it was possible to increase the accuracy from 0.906 with default parameters to 0.952 with optimized parameters with an execution time of 26 seconds. Table 4 and 5 show the confusion matrix with default values and with the EA, respectively.

		Predicted	
		bad	good
True	bad	27	9
	good	0	70

Table 4: Confusion matrix of ionosphere with standard parameters

		Predicted	
		bad	good
True	bad	35	3
	good	1	67

Table 5: Confusion matrix of ionosphere with EA

The handwritten digits data had a default accuracy of 0.925 and an optimized accuracy of 0.992. The algorithm required 10 minutes and 35 seconds for the 25 generations. The confusion matrix is shown in table 6 and 7.

		Predicted									
		0	1	2	3	4	5	6	7	8	9
	0	156	0	0	0	0	0	0	0	12	0
	1	0	155	0	0	0	0	0	0	19	2
	2	0	0	150	0	0	0	0	0	9	0
	3	0	0	0	155	0	0	0	0	15	2
	4	0	0	0	0	161	0	0	0	7	2
True	5	0	0	0	0	0	161	0	0	16	4
	6	0	0	0	0	0	0	155	0	20	0
	7	0	0	0	0	0	0	0	152	19	0
	8	0	0	0	0	0	0	0	0	166	0
	9	0	0	0	1	1	0	0	0	4	142

Table 6: Confusion matrix of handwritten digits with standard parameters

The third data set, the wall-following robot, had a default accuracy of 0.906 and an optimized accuracy of 0.952 with an execution time of 4 hours 12 minutes and 5 seconds. The confusion matrix can be found in table 8 and 9.

Furthermore, figure 4 and 5 show the development of the genotypes in terms of kernel function. In case of the ionosphere data set, the radial basis function prevails. With the handwritten digits, the polynomial function prevails.

		Predicted									
		0	1	2	3	4	5	6	7	8	9
	0	168	0	0	0	0	0	0	0	0	0
	1	0	174	0	0	0	0	0	0	1	1
	2	0	0	158	0	0	0	0	0	0	1
	3	0	0	0	171	0	0	0	0	0	1
	4	0	0	0	0	170	0	0	0	0	0
True	5	0	0	0	0	0	178	0	0	1	2
	6	0	1	0	0	0	0	174	0	0	0
	7	0	0	0	0	0	0	0	170	0	1
	8	0	3	0	1	0	0	1	0	160	1
	9	0	2	0	1	1	0	0	0	0	144

Table 7: Confusion matrix of handwritten digits with EA

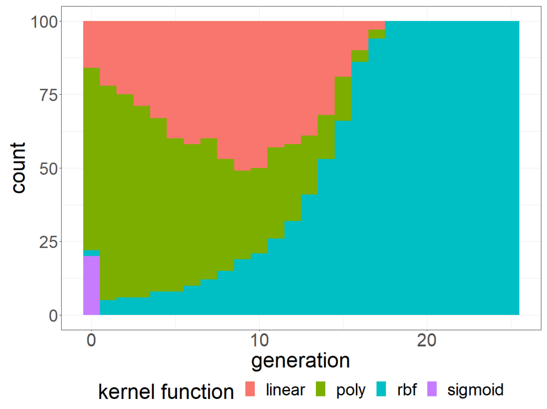


Figure 4: Development of the activation function: Ionosphere

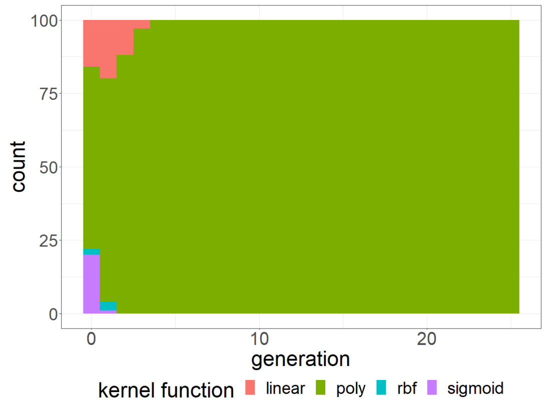


Figure 5: Development of the activation function: Handwritten Digits

The execution time of the ionosphere data set was faster than the others because the data set is significantly smaller in terms of number of rows, features and classes. The robot data set required more execution time than the handwritten digits despite being comparably large data sets. Generally according to the SVM documentation in scikit-learn, the underlying optimization method is a quadratic program solver whose runtime is in the range between $O(n_{features} \times n_{samples}^2)$ and $O(n_{features} \times n_{samples}^3)$. However for sparse data it is actually the number of non-zero features that play the crucial role and in contrast to the wall-following robot data set, the handwritten digits data set contains a fairly large number of zero-features on average which could explain the difference in execution time.

		Predicted			
		f	sr	l	r
True	f	385	34	2	12
	sr	23	393	0	4
	l	6	10	43	0
	r	31	8	0	141

Table 8: Confusion matrix of robot with standard parameters

		Predicted			
		f	sr	l	r
True	f	411	20	2	11
	sr	21	391	0	3
	l	4	2	62	0
	r	12	1	0	152

Table 9: Confusion matrix of robot with EA

7 DISCUSSION

For many observables, the running time is problematic because it grows exponentially. But even though the computation requirements of an EA are high, we can accept this due to the fact that the task is not time-critical [10].

As with any meta-heuristic, there is the problem of local maxima so the algorithm could get stuck in a possible good, but not optimal data point.

SVMs are not scale invariant, therefore it is usually recommended to scale the data, e.g. to [0,1] or [-1,+1] [11]. However, this makes reproducibility difficult.

The results of this paper show that a user can achieve a very good quality of results displayed in the 5 % increase of accuracy for all data sets. There are many benefits of this technique.

The solution proposed in this paper is quickly adaptable to all other possible models of the scikit-learn package as EA are a meta-heuristic optimization which is easy to use in combination with SVM. Furthermore, it is flexible and versatile since optimization of hyper-parameters is useful almost everywhere. Hyper-parameters are important because they have a direct influence on the training of a algorithm and impact significantly the performance of the trained model. Moreover, it is applicable to any data set.

Alternatively, grid search for the kernel learning could be applied, which is time-consuming.

Lastly, using other meta-heuristics and optimization methods (e.g. simulated annealing) for hyper-parameter tuning of SVMs and Machine Learning algorithms in general might be another interesting topic for further research. Evolutionary algorithms certainly constitute one of the more sophisticated meta-heuristic and it might be interesting to examine, whether similar results can be achieved with more simple ones. However, it is not obvious how to deal with the fact that different solution candidates might have different types and numbers of hyper-parameters as it is the case with all kernel based methods.

8 SUMMARY

Simple classifiers like support vector machines (SVM) can predict the class labels of unknown samples by learning from the training data. But the performance of SVM is sensitive to the selection of the adequate kernel function and the hyper parameter values associated with that kernel.

In this paper a support vector machine using an evolutionary algorithm was investigated to solve three different pattern classification problems. For the implementation, the scikit-learn class *sklearn.svm.SVC* was used to train the SVM and the evolutionary algorithm was implemented by ourselves in two classes using the object-oriented framework of python.

With the evolutionary approach it was possible to increase the accuracy of the SVM around 5 % for each data set compared to the default values of the SVM implemented in scikit-learn.

Our approach is especially useful for data sets, models and application purposes that do not need to be re-trained frequently.

Another interesting property of our approach is the possibility to gain insights about "regions" of hyper-parameters that correspond with good fitness values. This has already been touched upon in the visualization of the development of kernel functions over generations, however the potential applications are much more far-reaching. Especially in the case of algorithms which use hyper-parameters that are hard to grasp upon intuitively, by visualizing what constitutes good sets of hyper-parameters for specific problems, one can get a better understanding of the algorithm itself.

REFERENCES

[1] Hans-Georg Beyer and Hans-Paul Schwefel. 2002. Evolution strategies - A comprehensive introduction. *Natural Computing* 1 (03 2002), 3–52. https://doi.org/10.1023/A:1015059928466

[2] Christopher M. Bishop. 2016. *Pattern Recognition and Machine Learning.* Springer New York, Berlin-Heidelberg.

[3] Olivier Chapelle, Vladimir Vapnik, Olivier Bousquet, and Sayan Mukherjee. 2002. Choosing Multiple Parameters for Support Vector Machines. *Mach. Learn.* 46, 1–3 (March 2002), 131–159. https://doi.org/10.1023/A:1012450327387

[4] Dheeru Dua and Casey Graff. 2021. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml

[5] Frauke Friedrichs and Christian Igel. 2004. Evolutionary tuning of multiple SVM parameters. 519–524.

[6] Christian Igel. 2017. *Evolutionary Kernel Learning.* Springer US, Boston, MA, 465–469. https://doi.org/10.1007/978-1-4899-7687-1_93

[7] J. Jägersküpper and T. Storch. 2007. When the Plus Strategy Outperforms the Comma Strategy and When Not. *2007 IEEE Symposium on Foundations of Computational Intelligence* (2007), 25–32.

[8] Christian; Braune Christian; Klawonn Frank; Moewes Christian; Steinbrecher Matthias Kruse, Rudolf; Borgelt. 2015. *Computational Intelligence: Eine methodische Einführung in Künstliche Neuronale Netze, Evolutionäre Algorithmen, Fuzzy-Systeme und Bayes-Netze.* Springer-Verlag, Berlin, Heidelberg, New York.

[9] Volker Nissen. 1997. *Computational Intelligence: Einführung in Evolutionäre Algorithmen.* Vieweg+Teubner Verlag, Wiesbaden.

[10] Volker Nissen. 2017. *Applications of Evolutionary Algorithms to Management Problems.* https://doi.org/10.1007/978-3-319-64394-6_9

[11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[12] David G. Stork Richard O. Duda, Peter E. Hart. 2000. *Pattern Classification.* John Wiley and Sons Inc.