A Dynamic Early Stopping Criterion for Random Search in SVM Hyperparameter Optimization

Adrian Cătălin Florea, Răzvan Andonie

A Dynamic Early Stopping Criterion for Random Search in SVM Hyperparameter Optimization

Adrian Cătălin Florea[1] and Răzvan Andonie[2]

[1] Electronics and Computers Department, Transilvania University of Brașov, Brașov, Romania
acflorea@unitbv.ro
[2] Computer Science Department, Central Washington University, Ellensburg, WA, USA
andonie@cwu.edu

Abstract. We introduce a dynamic early stopping condition for Random Search optimization algorithms. We test our algorithm for SVM hyperparameter optimization for classification tasks, on six commonly used datasets. According to the experimental results, we reduce significantly the number of trials used. Since each trial requires a re-training of the SVM model, our method accelerates the RS optimization. The code runs on a multi-core system and we analyze the achieved scalability for an increasing number of cores.

1 Introduction

Most Machine Learning (ML) models are described by two sets of parameters. The first set consists in regular parameters that are learned through training. The other set, called *hyperparameters* or meta-parameters, consists of parameters which are set before the learning starts. It is essential to identify the combination of hyperparameter values which produce the best (or closed to the best) generalization performance. This is done by re-training multiple models with different combinations of hyperparameter values and evaluating their performance. We call this re-training + evaluation for one set of hyperparameter values a *trial*. Since training a model can be very resource intensive, it is important to reduce the number of trials.

In the specific case of SVM classifiers, the algorithm performance depends on several parameters and it is quite sensitive to changes in any of those parameters [1]. The choice of the *kernel*, for example, can have a dramatic influence on the classification performance [2]. The cost parameter (C), controlling the trade-off between margin maximization and error minimization is also highly important as, for the non-separable case, the algorithm must allow training errors. For a polynomial kernel, a wrong choice of the *degree* can easily lead to over-fitting [3].

The most commonly used hyperparameter optimization strategy is a combination of Grid Search (GS) and manual tuning[1] [4-6]. More elaborate techniques are: Nelder-Mead [7], simulated annealing [8], evolutionary algorithms [9], and Bayesian

[1] https://github.com/jaak-s/nips2014-survey - 82 out of 86 optimization related papers presented at the NIPS 2014 conference used GS

methods [10].

Random Search (RS) is another standard technique for hyperparameter optimization. A nice feature of RS is the possibility of adaptive early stopping. The key is to define a good stopping criterion, representing a trade-off between accuracy and computation time. The rise of the randomized methods begun with the work of Bergstra and Bengio [11,12]. Using the same number of trials, RS generally yields better results than GS or more complicated hyperparameter optimization methods. Especially in higher dimensional spaces, the computation resources required by RS methods are significantly lower than for GS [13]. Also, RS methods are relatively simple and easy to implement on parallel computer architectures.

Several software libraries dedicated to hyperparameter optimization exist, some of them being autonomous, while others being built on top of existing ML software. LIBSVM [14] and scikit-learn [15] come with their own implementation of GS, with scikit-learn also offering support for RS. Spearmint [16] and Bayesopt [17] are software packages dedicated to Bayesian optimization. Auto-WEKA [18] is also able to perform Bayesian optimization but, unlike the previous two which are standalone libraries, it is built on top of Weka [19]. Hyperopt [20] and Optunity [21] are currently two of the most advanced libraries for hyperparameter optimization.

Our contribution is an improved RS optimization technique, which reduces the number of trials, without a significant impact on the prediction performance. The key is a new dynamically calculated early stopping condition for RS. The method is implemented in parallel and achieves a good scalability. Our experiments are on the SVM classification problem applied to six commonly used datasets and five hyperparameters. According to them, our method accelerates the RS optimization.

The paper proceeds as follows. Section 2 describes our algorithm and the dynamic stopping condition, with an emphasis on the algorithm's parallel nature. Section 3 presents the experimental results and the paper is concluded with Section 4.

2 Proposed Algorithm and Probabilistic Properties

A highly simplified version of a hyperparameter optimization algorithm is characterized by an objective fitness function f and a generator of samples g. The fitness function returns a classification accuracy measure of the target model, computed either through cross-validation or on a separate validation set. The generator g is in charge of providing the next set of values that will be used to compute the model's fitness. A *hasNext* method implemented by the generator offers the possibility to terminate the algorithm before the maximum number of N evaluations is reached, if some convergence criteria is satisfied.

In the particular case of RS, the generator g simply draws samples from the specific distribution of each of the hyperparameters to be optimized. Our goal is to reduce the computational complexity of the RS method in terms of number of trials. In other words, we aim to compute less than N trials, without a significant impact on the value of the fitness function.

For this, we introduce a dynamic stopping criterion, included in a randomized

optimization algorithm (Algorithm 1). The algorithm is a two step optimizer. First, it iterates for a predefined number of steps n, $n << N$, and finds the optimal combination of hyperparameter values, *temp_opt*. Then, it searches for the first result better than *temp_opt*. The optimal result, *opt*, is either the first result better than *temp_opt* or *temp_opt* if N is reached.

Algorithm 1. Parametric stop optimizer

```
func Maximize(f, g, n, N){
      index = -1; tmp_opt = -math.MaxFloat64;  opt = tmp_opt
      for i := 0; i <= n; i++ {
            rndPoint := g.Next(); f_rnd, _ := f(rndPoint)
            if (f_rnd > tmp_opt) {
                index = i; p = rndPoint; tmp_opt = f_rnd
             }
      }
      for i := n+1; i < N; i++ {
             rndPoint := g.Next(); f_rnd, _ := f(rndPoint)
            if (f_rnd > tmp_opt) {
                index = i; p = rndPoint; opt = f_rnd
                break
            }
      }
      return index, p, opt
}
```

The following problems arise: *i)* Can we determine a value for n that maximizes the probability of obtaining the best results?; and *ii)* Can the algorithm be parallelized without impacting the probability of obtaining an optimal value?

2.1. Sequential execution

Algorithm 1 finds the optimum under the assumption that *opt* is in any position i, $i > n$, and no result better than *temp_opt* is in the range $[n+1, i-1]$.

We denote by $E1_i$ the event that *opt* is reached on the i-th trial, and by $E2_i$ the event that no value better than *temp_opt* is obtained between the n-th and the i-th trial. The probability of $E1_i$ is

$$P(E1_i) = 1/N \tag{1}$$

The probability that all values in the range $[n+1, i-1]$ are worse than *temp_opt* is the same as the probability that the best result among the first $i-1$ attempts lays in the range $[1, n]$:

$$P(E2_i) = n/(i-1) \tag{2}$$

Since the two events are independent, the probability that we hit *opt* after $i > n$ attempts is:

$$P_i = P(E1_i) \cdot P(E2_i) = n/(N(i-1)) \tag{3}$$

The event E of finding *opt* after at most m trials, $n < m < N$ (where $m = N/2$ is a reasonable target), has probability

$$P(E) = \sum_{i=n+1}^{m} P_i = \frac{n}{N} \sum_{i=n+1}^{m} \frac{1}{(i-1)} = \frac{n}{N} \sum_{i=n}^{m-1} \frac{1}{i} \tag{4}$$

Since $1/i$ is monotonically decreasing, the right term of eq. (4) has a lower bound:

$$\frac{n}{N} \int_{n}^{m} \frac{1}{x} dx \leq \frac{n}{N} \sum_{i=n}^{m-1} \frac{1}{i} \tag{5}$$

We differentiate the left term of eq. (5):

$$\frac{d}{dn}\left(\frac{n}{N} \left(\ln m - \ln n \right) \right) = \frac{1}{N} (\ln m - \ln n - 1), \tag{6}$$

equate to zero and solve for n obtaining:

$$n = m/e \tag{7}$$

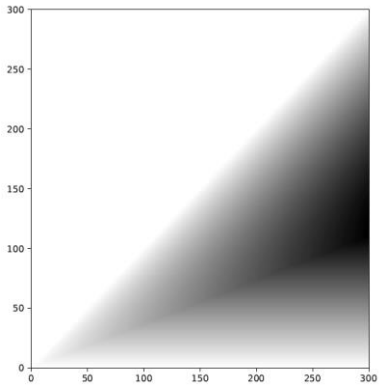

Fig.1. Lower bound heatmap of the probability to obtain best result from a target space of maximum 300 attempts while terminating faster, depending on the values of m (x axis) and n (y axis). Darker shades correspond to greater probability.

Choosing for n a value larger than the optimal one increases the probability of finding the combination of values that yields the optimal result but with an increased risk of a greater number of trials. The result from eq. (7) can be used to implement an improved version of the Algorithm 1 that can automatically set the value of n to N/e. For example, in order to maximize the chances to obtain the best value, after a target maximum of 150 attempts, we must set n to $150/e$ (≈ 55). For a target maximum of 100 attempts, n should be 37, and so on. Fig. 1 shows the lower bound heatmap of the probability to obtain the best results while stopping earlier with respect to the values of m and n.

2.2. Parallel execution

We generate a parallel implementation of our method as follows:

- Split the work between W workers (can be anything from lightweight threads of execution, OS threads, CPU cores or even different servers). We

decided to use the GOLANG [22] support for goroutines[2], which are basically lightweight threads managed by the GO run-time.

- Each worker w executes N/W trials using the same early stopping criterion. In this case, $n_W = n/W$, signifying that on average, with $m = N/2$, W workers will terminate after $N/(2W)$ trials, with N/W being the worst case.
- The manager gathers the results from all workers and selects the best candidate.

Algorithm 2 implements the above steps. The random values are generated and distributed by the manager, or each worker generates its own random sequence. Any of the following parallel pseudo-random number generation strategies can be selected [23]: Manager-Worker (MW), Sequence Splitting (SS), Leapfrog (LF), and Parametrization (P).

Algorithm 2 Parallel stop optimizer

```
func PMaximize(f, g, N, W){
     // channels used by workers to communicate their results
     resultsChans := make(chan fn.Sample, W)
     for w := 0; w < W; w++ {
          go func(w int) {
               // index, point, function value, global optim, k
               i, p, v, gv, k := Maximize(f, g, N/W, w )
               resultsChans <- fn.Sample{i, p, v, gv, k}
          }(w)
     }
     // Collect results
     results := make([]fn.Sample, W)
     for i := 0; i < W; i++ {
          results[i] = <-resultsChans
     }
}
```

2.3. The inverse problem

Given a restricted computational budget, expressed by a target number of trials m, we obtained the optimal value for n. We are now interested in solving the reverse problem: Given an acceptable probability P to achieve the best result among the N trials, which is the optimal value for n?

For the RS algorithm without the dynamic stopping criterion, if all trials are independent, the required number of trials needed to identify the optimum with a probability P_0 is given by $m = N \cdot P_0$ The problem becomes interesting in the context of our stopping criterion when we are willing to compromise, by accepting a lower probability $P < P_0$, for a further reduction of the number of trials.

In case of Algorithm 2, according to eq. (5), probability P has a lower bound:

$$P \geq \frac{n}{N}\left(1 + \int_{n}^{m} \frac{1}{x}\,dx\right) = \frac{n}{N} \ln \frac{em}{n} \tag{8}$$

[2]https://tour.golang.org/concurrency/1

This, together with eq. (7) gives:

$$P \geq= m/(eN) \cdot \ln e^2 = 2P_0/e \approx 0.7357P_0 \quad (9)$$

The value from eq. (9) represents the probability to identify the optimum regardless of the activation of the stopping criterion - opt might also be among the first n trials in which case the algorithm will test all the N possible combinations. The probability to find the optimum after a number of trials strictly lower than N has a lower bound given by relation (5), which translates to:

$$P \geq m/(eN) = P_0/e \approx 0.3678P_0 \quad (10)$$

The value of n in eq. (8) can be adjusted in the interval $[m/e, m]$ to increase the probability of identifying the optimal value, but at the same time increase the computational cost (the number of trials).

3 Experiments

We use our method to optimize the following five hyperparameters of a SVM [1] classifier: kernel type (RBF, Polynomial or Linear chosen with equal probability), γ (drawn from an exponential distribution with $\lambda = 10$); cost(C, drawn from an exponential distribution with $\lambda = 10$); degree (chosen with equal probability from the set $\{2,3,4,5\}$) and $coef0$ (uniform on $[0,1]$).

We run our experiments on six of the most popular datasets from UCI Machine Learning Repository [3]: Adult (a1a), Adult (a6a), Breast Cancer, Diabetes, Iris and Wine. Adult (a1a) and Adult (a6a) are variations of the same dataset but with different number of samples; the second one is around six times larger. Details of the datasets are presented in Table 1.

Table 1. Details on used datasets

Dataset	Instances	Features	Classes #
Adult (a1a)	1,605	123	2
Adult (a6a)	11,220	123	2
Breast Cancer	683	10	2
Diabetes	768	8	2
Iris	150	4	3
Wine	178	13	3

We apply ten fold cross-validation to evaluate the classification accuracy [24] and compare the obtained results, both in terms of classification performance and number of trials. We use the following optimizers (all implemented in the Optunity library): GS, RS, Particle Swarm, and Nelder-Mead. We also use the Weka SVM, with its implicit hyperparameters.

We run the Algorithm 2 with $W = 8$ and $N = 250$, which leads to $n = 92$. We also run the four optimizers in Optunity, for a maximum number of 250 trials.

[3] http://archive.ics.uci.edu/ml/index.php

3.1. Accuracy estimation

Table 2 presents the results of applying Algorithm 2 for four parallelization strategies, compared with the results obtained with Optunity (RS, GS, Particle Swarm and Nelder-Mead) as well as with the results obtained using Weka with each of the three kernels (RBF, Polynomial and Linear) and the implicit values for the other parameters ($C = 1.0$, $\gamma = 1/number\_of\_features$, $degree = 3$, $coef0 = 0.0$). The best results are marked in bold.

Table 2. Accuracy and number of trials for Algorithm 2 using different parallelization strategies (MW, SS, LF, P), compared with Optunity (RS, GS, Particle Swarm and Nelder-Mead) and Weka's SVM.

	GO MW		GO SS		GO LF		GO P		Optunity RS	Optunity GS	Optunity PS	Optunity NM		Weka RBF	Weka Poly	Weka Linear
Dataset	**Acc**	**Runs**	**Acc**	**Runs**	**Acc**	**Runs**	**Acc**	**Runs**	**Acc**	**Acc**	**Acc**	**Acc**	**Runs**	**Acc**	**Acc**	**Acc**
Adult (a1a)	0.837	164	0.836	194	0.837	194	0.837	148	0.837	0.835	0.838	0.833	108	0.828	**0.839**	0.754
Adult (a6a)	0.844	169	0.844	133	0.844	203	**0.845**	138	0.844	0.844	0.845	0.843	142	0.838	0.760	0.760
Cancer	0.975	172	0.975	167	0.975	214	0.975	187	0.975	0.975	**0.975**	0.968	17	0.969	0.974	0.969
Diabetes	0.776	223	0.776	203	**0.780**	220	0.779	130	0.777	0.777	0.776	0.651	6	0.775	0.686	0.777
Iris	0.973	224	0.980	156	0.980	189	0.973	158	0.980	**0.987**	0.967	0.940	6	0.953	0.727	0.960
Wine	**0.989**	194	**0.989**	215	**0.989**	162	**0.989**	177	**0.989**	0.984	**0.989**	0.956	62	0.983	0.404	0.972
Average	0.899	191	0.900	178	**0.901**	197	0.900	156.334	0.900	0.900	0.898	0.865	**56.834**	0.891	0.732	0.865
STDEV	0.091	27.188	0.092	31.241	0.091	**20.746**	0.090	22.223	0.092	0.092	0.090	0.120	57.711	**0.088**	0.190	0.112

Since we compare multiple classifiers on multiple datasets, we have to use additional statistical tests for further investigation, as suggested in [25].

We calculate the Friedman [26] and the Iman-Davenport [27] statistics using eq. (11), respectively eq. (12), with N being the number of datasets, k the number of algorithms and R_j the average rank of algorithm j from Table 3, and obtain $\chi_F^2 = 32.826, F_F = 6.04$.

$$\chi_F^2 = \frac{12N}{k(k+1)}\left[\sum_{j=1}^{k} R_j^2 - \frac{k(k+1)^2}{4}\right] \tag{11}$$

$$F_F = \frac{(N-1)\chi_F^2}{N(k-1)-\chi_F^2} \tag{12}$$

With 11 algorithms and six data sets, F_F is distributed according to the F distribution with $11-1=10$ and $(11-1)\times(6-1)=50$ degrees of freedom. The critical value of $F(10,50)$ for $\alpha = 0.05$ is 2.03, so we reject the null-hypothesis, which means the algorithms are not equivalent in terms of prediction performance.

The critical difference [28,25] is given by:

$$CD_\alpha = q_\alpha \sqrt{\frac{k(k+1)}{6N}} \tag{13}$$

where critical values q_α are based on the Studentized range statistic divided by $\sqrt{2}$. At significance level of $\alpha = 0.05$, the critical difference is $CD_{0.05} = 6.163$.

Table 3. Algorithms' accuracy ranking on the used datasets.

Dataset	GO MW	GO SS	GO LF	GO P	Optunity RS	Optunity GS	Optunity PS	Optunity NM	Weka RBF	Weka Poly	Weka Linear
Adult (a1a)	6	7	3.5	3.5	5	8	2	9	10	1	11
Adult (a6a)	3	4	7	1	5	6	2	8	9	10.5	10.5
Cancer	5.5	5.5	5.5	5.5	2.5	2.5	1	11	9.5	8	9.5
Diabetes	8	7	1	2	5	4	6	11	9	10	3
Iris	5.5	3	3	5.5	3	1	7	10	9	11	8
Wine	2.5	2.5	2.5	2.5	5.5	7	5.5	10	8	11	9
Average	5.083	4.833	3.750	3.333	4.333	4.750	3.917	9.833	9.083	8.583	8.500

This clearly rules out the Nelder-Mead algorithm, which is significantly worse than the Parametrization based implementation of our RS ($9.833 - 3.333 > 6.163$). At significance level $\alpha = 0.1$, the critical difference is $CD_{0.1} = 5.701$ and we observe that Optunity - NM is significantly worse than GO - LF and GO - P and also that GO - P is significantly better than Optunity - NM and Weka - RBF.

3.2. Efficiency of the stopping condition

Based on the above results, we exclude Nelder-Mead (due to its significantly worse classification performance) and the Weka SVM (since is it does not perform a real hyperparameter optimization) from the analysis. Table 4 depicts the rank across all datasets in terms of number of trials.

We perform another Friedman test and, using formulas (11) and (12), and obtain: $\chi_F^2 = 28.554$ and $F_F = 19.173$. The critical value for $F(6,30)$ ($7-1$ and respectively $(7-1)(6-1)$) is 2.420. This means that we can rule out the null-hypothesis and state that the algorithms are not equivalent with respect to the number of trials. We compute the critical difference according to formula (13) and obtain $CD_{0.05} = 3.678$.

Table 5 shows the difference in the average rank values for each pair of algorithms. The values greater than $CD_{0.05}$ are marked in bold font. We can identify two groups of algorithms, the first group (GO - SS and GO - P) performs significantly better than the second group (Optunity - GS, Optunity - RS and Optunity - PS). It is not clear to which of the two groups GO - MW and GO - LF belong to. One possible explanation for the better results obtained by GO - SS and GO - P may be related to the superior parallel implementation of the random generators. However, since the number of random values generated in our tests is relatively small, this difference in

performance is most probably coincidental.

Table 4. Algorithms ranking in terms of number of runs.

Dataset	GO MW	GO SS	GO LF	GO P	Optunity RS	Optunity GS	Optunity PS
Adult (a1a)	2	3.5	3.5	1	6	6	6
Adult (a6a)	3	1	4	2	6	6	6
Cancer	2	1	4	3	6	6	6
Diabetes	4	2	3	1	6	6	6
Iris	4	1	3	2	6	6	6
Wine	3	4	1	2	6	6	6
Average	3.000	2.083	3.083	1.833	6.000	6.000	6.000

Table 5. Algorithms ranking difference in terms of number of runs.

	GO MW	GO SS	GO LF	GO P	Optunity RS	Optunity GS	Optunity PS
GO MW	-	-0.917	0.084	-1.167	3	3	3
GO SS		-	1	-0.25	**3.917**	**3.917**	**3.917**
GO LF			-	-1.25	2.917	2.917	2.917
GO P				-	**4.167**	**4.167**	**4.167**
Optunity RS					-	0	0
Optunity GS						-	0
Optunity PS							-

Finally, since the main goal of our work is to obtain an improved version of RS, we compare our method directly with Optunity RS, using the Holm [29] test. The standard error for our experiment is $SE = \sqrt{k(k+1)/(6N)} = 1.247$. Table 6 shows the results of the Holm rejection test.

Table 6. Performance in terms of number of trials required for GO - MW, GO - P, GO - SS and GO - LF against Optunity - RS in terms of the Holm test.

	i	$z = (R0 - Ri)/SE$	p	α/i
GO P	1	3.341	0.00084	0.0084
GO SS	2	3.140	0.00043	0.01
GO MW	3	2.405	0.00808	0.0125
GO LF	4	2.339	0.009668	0.0167

The Holm test rejects all four hypotheses, since the corresponding p values are smaller than the adjusted α's, leading to the conclusion that all four versions of our algorithm are significantly more efficient in terms of number of trials than the standard RS implementation.

3.3. Scalability

Besides the accuracy and the number of runs we also measure the algorithm's speedup (the ratio of the sequential execution time to the parallel execution time) as a

measure of its scalability. The values are depicted in Table 7.

Table 7. Algorithm speedup with increasing number of cores.

Dataset/Cores	2	3	4	6	8
Adult(a1a)	1.37	2.96	3.48	4.13	4.52
Adult (a6a)	1.97	2.71	3.02	3.35	3.70
Cancer	1.98	2.91	3.53	3.91	4.09
Diabetes	1.86	2.72	3.34	3.70	3.88
Iris	1.94	2.75	3.10	3.43	3.54
Wine	1.99	2.81	3.28	3.96	4.11
Average	1.85	2.81	3.29	3.75	3.97

4 Conclusions

We introduced a new dynamic stopping condition for RS based hyperparameter optimization, together with its parallel implementation. In the context of SVM classification, on six of the most commonly used datasets, we obtained on par accuracy values with the existing mainstream hyperparameter optimization techniques. With all four of the parallel random generators used, the algorithm terminates after a significantly reduced number of trials compared to the standard implementation of RS, which leads to an important decrease in the computational budget required for the optimization.

The present work opens further research directions in terms of optimizing the hyperparameters for other ML algorithms where the search space has a larger number of dimensions and the required computational budget is currently a major issue. The algorithm implementation is flexible enough to allow a gradient-free optimization of any function.

References

1. C. Cortes and V. Vapnik, “Support-vector networks,” Mach. Learn., vol. 20, no. 3, pp. 273–297, Sep. 1995. [Online]. Available: http://dx.doi.org/10.1023/A:1022627411411
2. O. Chapelle, V. Vapnik, O. Bousquet, and S. Mukherjee, “Choosing multiple parameters for support vector machines,” Machine Learning, vol. 46, no. 1, pp. 131–159, Jan 2002. [Online]. Available: https://doi.org/10.1023/A:1012450327387
3. C. M. Bishop, Pattern Recognition and Machine Learning (Information Science and Statistics). Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.
4. Y. LeCun, L. Bottou, G. Orr, and K. Mller, Efficient Backprop, ser. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Springer, 2012, vol. 7700 LECTURE NO, pp. 9–48.
5. G. E. Hinton, “A practical guide to training Restricted Boltzmann Machines.” in Neural Networks: Tricks of the Trade (2nd ed.), ser. Lecture Notes in Computer Science, G. Montavon, G. B. Orr, and K.-R. Mller, Eds. Springer, 2012, vol. 7700, pp. 599–619.

6. S. Smusz, W. M. Czarnecki, D. Warszycki, and A. J. Bojarski, “Exploiting uncertainty measures in compounds activity prediction using support vector machines,” Bioorganic & medicinal chemistry letters, vol. 25, no. 1, pp. 100–105, 2015.
7. J. A. Nelder and R. Mead, “A Simplex Method for Function Minimization,” Computer Journal, vol. 7, pp. 308–313, 1965.
8. S.Kirkpatrick,“Optimization by simulated annealing:Quantitative studies,”Journal of Statistical Physics, vol. 34, no. 5, pp. 975–986, Mar 1984.
9. N. Hansen, S. D. Muller, and P. Koumoutsakos, “Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es),” Evol. Comput., vol. 11, no. 1, pp. 1–18, Mar. 2003. [Online]. Available: http://dx.doi.org/10.1162/106365603321828970
10. C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, “Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms,” in Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ser. KDD ’13. New York, NY, USA: ACM, 2013, pp. 847–855. [Online]. Available: http://doi.acm.org/10.1145/2487575.2487629
11. J. Bergstra, R. Bardenet, Y. Bengio, and B. Kgl, “Algorithms for hyper-parameter optimization.” in NIPS, J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. C. N. Pereira, and K. Q. Weinberger, Eds., 2011, pp. 2546–2554. [Online]. Available: http://dblp.uni-trier.de/db/conf/nips/nips2011.html
12. J. Bergstra and Y. Bengio, “Random Search for Hyper-Parameter Optimization,” Journal of Machine Learning Research, vol. 13, pp. 281–305, 2012.
13. J. Lemley, F. Jagodzinski, and R. Andonie, “Big holes in big data: A monte carlo algorithm for detecting large hyper-rectangles in high dimensional data,” in 2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC), vol. 1, June 2016, pp. 563–571.
14. C.-C. Chang and C.-J. Lin, “LIBSVM: A library for support vector machines,” ACM Transactions on Intelligent Systems and Technology, vol. 2, pp. 27:1–27:27, 2011, software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.
15. 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, “Scikit-learn: Machine learning in Python,” Journal of Machine Learning Research, vol. 12, pp. 2825–2830, 2011.
16. J. Snoek, H. Larochelle, and R. P. Adams, “Practical bayesian optimization of machine learning algorithms,” in Advances in Neural Information Processing Systems 25, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 2951– 2959. [Online]. Available: http://papers.nips.cc/paper/4522-practical-bayesian-optimization-of-machine-learning-algorithms.pdf
17. R. Martinez-Cantin, “Bayesopt: A bayesian optimization library for nonlinear optimization, experimental design and bandits,” CoRR, vol. abs/1405.7430, 2014. [Online]. Available: http://arxiv.org/abs/1405.7430
18. L. Kotthoff, C. Thornton, H. H. Hoos, F. Hutter, and K. Leyton-Brown, “Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA,” Journal of Machine Learning Research, vol. 18, no. 25, pp. 1–5, 2017. [Online]. Available: http://jmlr.org/papers/v18/16-261.html

19. M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: An update," SIGKDD Explor. Newsl., vol. 11, no. 1, pp. 10–18, Nov. 2009. [Online]. Available: http://doi.acm.org/10.1145/1656274.1656278
20. J. Bergstra, B. Komer, C. Eliasmith, D. Yamins, and D. D. Cox, "Hyperopt: a Python library for model selection and hyperparameter optimization," Computational Science and Discovery, vol. 8, no. 1, p. 014008, 2015. [Online]. Available: http://stacks.iop.org/1749-4699/8/i=1/a=014008
21. M. Claesen, J. Simm, D. Popovic, Y. Moreau, and B. D. Moor, "Easy Hyperparameter Search Using Optunity," CoRR, vol. abs/1412.1114, 2014. [Online]. Available: http://arxiv.org/abs/1412.1114
22. Google. (2007) The Go programming language. [Online]. Available: https://golang.org/pro ject/
23. M. J. Quinn, Parallel Programming in C with MPI and OpenMP. McGraw-Hill Education Group, 2003.
24. M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," Inf. Process. Manage., vol. 45, no. 4, pp. 427–437, Jul. 2009. [Online]. Available: http://dx.doi.org/10.1016/j.ipm.2009.03.002
25. J. Demsar, "Statistical comparisons of classifiers over multiple data sets," Journal of Machine Learning Research, vol. 7, pp. 1–30, 2006. [Online]. Available: http://www.jmlr.org/papers/volume7/demsar06a/demsar06a.pdf
26. M. Friedman, "A comparison of alternative tests of significance for the problem of m rankings," Ann. Math. Statist., vol. 11, no. 1, pp. 86–92, 03 1940. [Online]. Available: http://dx.doi.org/10.1214/aoms/1177731944
27. R. Iman and J. Davenport, "Approximations of the critical region of the Friedman statistic," Communications in Statistics-Theory and Methods, vol. 9, pp. 571–595, 01 1980.
28. P. Nemenyi, Distribution-free Multiple Comparisons. Thesis Princeton University, 1963. [Online]. Available: https://books.google.ro/books?id=nhDMtgAACAAJ
29. S. Holm, "A simple sequentially rejective multiple test procedure," Scandinavian Journal of Statistics, vol. 6, pp. 65–70, 1979.