SEE-Classify: Simple Evolutionary Exploration Tool to Search Classifiers and their Hyper-parameters

Anonymous Author(s)
Affiliation
Address
email

Abstract

Genetic algorithms (GAs) find solutions to search problems through a process inspired by evolution. Possible solutions to a problem are randomly selected and tested using a fitness function. The best solutions undergo changes (mutations) over multiple iterations (generations) to try and find better solutions. There have been several studies that use GAs to search over hyperparameters of machine learning algorithms to learn values that work well for specific problems. In this work an existing GA framework was extended to search over different classifiers and their hyperparameters. This will allow scientists from any field to search a classifier "algorithm space" to find a specific classifier (Support Vector Machine, Forest of Decision trees, Neural Networks, etc.) that works well for their problem. This paper demonstrates the feasibility of the SEE-Classify system by testing the system on well-known classification examples provided with the Scikit-learn Python Library and reproducing results from a previous study that performs simlar hyper-parameter genetic search over diagnostic breast cancer data.

1 Introduction

Posed as a question, the Algorithm Selection Problem asks: “Which algorithm is likely to perform best for my problem?” [11],[12]. Given a plethora of algorithms where no single choice is the best for all problems under a certain domain, choosing an algorithm for a specific problem is a nontrivial task. In the context of machine learning algorithms, hyperparameters further increase the complexity of algorithm selection. This work defines hyperparameters as the set of values that must be specified by the user to control the learning process of the algorithm before training begins. Hyperparameter tuning is one method to help maximize the performance of a machine learning algorithm. As there is no existing analytic approach to selecting hyperparameters a priori, common strategies include but are not limited to choosing manually via rules of thumb, testing a predefined set of hyperparameter values, or random search [4]. The search space of algorithm selection increases rapidly with each additional algorithm, which may bring its own distinct set of hyperparameters. For example, a search space of only 5 algorithms, each with 5 distinct parameters, which each can range over 100 possible values consists of 5 * 5 * 100 = 2500 distinct choices. If each algorithm takes an average of 3 seconds to train and test a classifier searching the entire space using brute force would take 2 hours. Although this is reasonable for many small problems adding additional algorithms, parameters and training data can quickly scale the problem so that it becomes intractable.

Algorithm selection is a non-trivial problem that exists across disciplines and is difficult to solve by manual search without expert knowledge. The motivation for the SEE tools [2], is to address algorithm selection within the domain of scientific image analysis by using a simple Genetic Algorithm (GA) to traverse the search spaces over specific data analysis workflows. GAs are popularly used techniques for search and optimization problems. It has been used in several studies to search a space of machine learning classification algorithms. As such it would be appropriate to apply the SEE toolkit and framework onto supervised-learning classification algorithms. The goal of this project is to extend the SEE toolkit to support classification algorithms via a see-classify module as well as provide a proof of concept by comparing the performance of see-classify with a previous work of hyperparameter tuning via genetic search [5]. This work utilizes the machine learning algorithms implemented in the Scikit-learn package [10].

2 Related works

Machine learning models are pipelines that consist in typical stages of pre-processing, feature extraction, and applying the machine learning algorithm. There have been several studies that use Genetic Search techniques such as Genetic Programming (GP) and Genetic Algorithms (GAs) to improve or create high-quality machine learning models. Dhahri et al. used GP to find high quality combinations of each stage of the machine learning model [5]. Ferreira, et al used GP to improve the interpretability of specific black-box classification algorithms [7]. Wicaksono and Afif used GA to tune the hyperparameters of specific machine learning algorithms [13]. There has also been work done for using genetic algorithms to tune the hyper-parameters or improve the search of specific classification algorithms such as neural networks [8],[9],[15].

Tackling the Algorithm Selection Problem falls under the larger category of Meta-Learning which has become an important field of Machine Learning. While there are many definitions of meta-learning, in this work meta-learning is defined as learning about learning algorithm performance. This work uses a simple Genetic Algorithm to compare and search for a good Classification algorithm for a dataset. As such it falls under the category of meta-learning.

This work focuses on providing software tools that can be used by others without having to recreate the Genetic Algorithm machinery as well as comparing the application of Genetic Algorithms with the existing literature as a form of quality-assurance. This is meant to be a general purpose tool that can be reused by scientists from any domain. This work also investigates the rate at which our GA-based system can converge with results found by the existing literature as well as compare the quality of the solutions found by each system.

3 Proposed method

3.1 Experimental approach

This work performs two proofs of concept using a simple GA. The first replicates a demo example from the scikit-learn website [1] which consists allegorically generated data-sets including: moons, circles, and linearly separable. With the assumption that the algorithms used in the tutorial were manually tuned and selected, the algorithm search space is defined using those algorithms and some of their parameters. The second attempts to replicate a prior work on hyperparameter tuning and algorithm selection via Genetic Programming [5] which uses the Wisconsin Breast Cancer Diagnostic (WBCD) dataset [14] and can be found in the UCI Machine Learning Repository [6]. A summary of these datasets can be found in Table 1.

Table 1: Dataset characteristics

	Moons	Circles	Linearly separable	WBCD
# items	100	100	100	569
# features	2	2	2	30
# classes	2	2	2	2

Genetic Algorithm (GA) The basis of this work is a simple genetic algorithm. Algorithms are represented as lists of hyperparameter values, where all algorithm consists of the same aggregate set of hyperparameters. For our experiments, these lists are converted into real machine learning models by using the corresponding algorithm and hyperparameters implemented in the scikit-learn package. Not all hyperparameters are used for each specific algorithm and some parameters such as max_depth and learning_rate are shared by several algorithms. Reducing this aggregate set of

Table 2: Algorithm space

Name	Hyperparameters
Ada Boost	learning_rate, n_estimators
Decision Tree	max_depth
Extra Trees	max_depth, n_estimators
Gaussian Naive Bayes	var_smoothing
Gaussian Process	
Gradient Boosting	learning_rate, n_estimators
K Nearest Neighbors	n_neighbors
Linear Discriminant Analysis	
Logistic Regression	C, max_iter
Neural Network	activation, alpha, max_iter, solver
Quadratic Discriminant Analysis	
Random Forest	max_depth, n_estimators
SVC	C, gamma, kernel

parameters in a list is an area of future work. Cross-over is performed by swapping random sections of each algorithm list. Mutation is performed by regenerating a hyperparameter value from the range of possible values for that parameter. In the simple GA, the first population is randomly generated. The ten best solutions of are selected via the fitness function and cross-over and mutation is performed over them to create one portion of the population of the next generation. The remaining portion of the population is randomly generated. This process of selection, cross-over, mutation, and random generation continues in a loop and GA terminates after reaching a specified number of iterations (i.e. "generations").

Parameter search space The algorithms included in SEE-classify software are listed in Table 2. However, the algorithms explored in the Scikit-learn tutorial are slightly different from those explored in Dhahri et al. [5]. For example, the tutorial did not include Extra Trees or Logistic Regression which were used in latter. In an effort to be complete the search space for SEE-Classify includes all of the algorithms (listed in Table 2, with their corresponding hyperparameters) used in both sources. However, experiments were conducted with subsets of the algorithm spaces in order better compare the results with the prior work and demonstrate the flexibility of the software and GA approach. These subsets were chosen such that they match and correspond to the algorithm spaces used in Scikit-learn tutorial and the Dhahri et al. Table 3 lists the two algorithm space subsets used in each replication experiment.

Table 4 lists the hyperparameters implemented in the software. The transformation column for each hyperparameter shows how the corresponding input range is transformed into a range of possible values that would be used for machine learning algorithms. The transformation column specifies how the input range is transformed before being used in a machine learning algorithm. For example, the hyperparameter *C* has an input range from 0 to 6 (inclusive) which step-sizes of 1. The corresponding transformation is 10^x, which means that internal to SEE-Classify the range of possible hyperparameter values for *C* are: 1, 10, 100, 1000, 10000, 1000000. This work specifies the input ranges, step-size, and transformations such that the default hyperparameter value for each algorithm (as specified in scikit-learn) is included in the range of possible hyperparameter values (after the transformation is applied). Where the transformation is not specified, the corresponding input range is used directly as this range. In particular, the "gamma" hyperparameter can be either a string or non-integer. To include both possible types of values, this work specifies both the transformation that begets the possible non-integer values as well as the explicitly specify all possible string values. The string values are listed after the transformation itself.

Fitness function This work define the normalized fitness function as the ratio: $\frac{\text{\# incorrect labels}}{\text{total \# of items in dataset}}$. A solution with a fitness value of zero (0) has labelled all the items correctly; whereas one with a fitness value of one (1) will have labelled all the items incorrectly. This paper acknowledges that any fitness function that it pick will have some form of bias. For example datasets where there are distinct majority and minority subsets of data of different sizes, the GA might prefer solutions that

Table 3: Replication-specific search spaces

Algorithm search space	Scikit-learn Tutorial	Dhahri et al. 2019
Ada Boost	✓	✓
Decision Tree	✓	✓
Extra Trees		✓
Gaussian Naive Bayes	✓	✓
Gaussian Process	✓	
Gradient Boosting		✓
K Nearest Neighbors	✓	✓
Linear Discriminant Analysis		✓
Logistic Regression		✓
Neural Networks	✓	
Quadratic Discriminant Analysis	✓	✓
Random Forest	✓	✓
SVC	✓	✓

Table 4: Hyperparameter space

Name	Type	Input Range; Step-size	Transformation
activation	string	["identity", "tanh", "logistic", "relu"]	
alpha	numeric	[-6, -1]; 1	10^x
C	numeric	[0, 6]; 1	10^x
gamma	numeric, string	[-6, 6], 1	10^x; 'scale', 'auto'
kernel	string	["linear", "poly", "rbf", "sigmoid"]	
max_depth	integer	[1, 30]; 1	
max_iter	integer	[200, 1000]; 100	
n_estimators	integer	[50, 1000]; 50	
n_neighbors	integer	[1, 30]; 1	
learning_rate	numeric	[-6, 0]; 1	10^x
solver	string	["lbfgs", "sgd", "adam"]	
var_smoothing	numeric	[-18, 18]; 1	10^x

can more correctly label labels in the larger subsets. This is problematic as the classifier may be systematically inaccurate when classifying particular minority subsets. Additionally, when finding research to replicate, it was noticed that the literature uses several different fitness functions. This serves as another challenge that makes true comparison difficult. However, this function was chosen for its simplicity and plan to explore alternative fitness functions in the future. The SEE-Classify tool is designed such that it should be relatively simple to swap out fitness functions which will be an area of future work.

3.2 Data collection and challenges

All experiments were run on the Michigan State University (MSU) High Performance Computer Center (HPCC) utilizing job arrays with different input seeds on cores across the cluster. In these experiments each core consisted of an Intel(R) Xeon(R) CPU E5-2670 v2 @ 2.50GHz.

3.2.1 Scikit-learn tutorial

The scikit-learn tutorial [1] uses the three toy datasets: moons, circles, and linearly separable, which are respectively generated using the following functions: make_moons, make_circles, make_classification and applying noise with particular scales. The noise was randomly generated using specific random states (i.e. known random seeds) to ensure reproducibility. A 60-40 split was used to generate the training and testing sets. To use SEE-Classify to replicate the problem, the same 60-40 split is applied during Genetic Search. After the GA ends, the performance of the best found solutions are evaluated by fitting them on the entire dataset and scoring them using a series of validation sets, which are created by regenerating the noise applied to each dataset on different random states. Validation is performed on a different set of generated data to check whether the found models can generalize well. This experiment ran the GA for 100 different trials using a population size of 100 for 100 generations (more than sufficient for these toy examples). The experiment was not optimized for time and was run using the MSU HPCC in parallel on 75 independent cores. Each core took between 2 to 7 hours to run, for a total of approximately 294 hours and 36 minutes of runtime for all of the experiments.

One of the challenges faced was deciding on the data split. A 60-40 split was specifically chosen so that the fitness score can have values of 2.5% increments. This has two benefits. First, the GA would be able to find better solutions than had more common splits like 80-20 or 70-30 been used. This is because an 80-20 or 70-30 split would limit the fitness function to 5% and 3% increments meaning that the best fitness score that could be achieved is either 0 or the value of the respective increments. This makes it impossible to detect the minor nuances models that could have had a fitness value of 2.5 which would have been lost when using either of the more common splits. Second, using the same training and testing sets as the tutorial would eliminate the effect that using a different testing set might have on the GA.

Another challenge was regenerating the machine-learning models for validation. The GA framework that is extended to create SEE-Classify makes it difficult to preserve the model that was trained and tested during genetic search. One of the big assumptions that was made is that retraining using the same hyperparameters would yield a similar model, which is not always true. However, doing so is easier given the existing framework. Additionally, because the models need to be retrained the training *and* testing subsets were used for training rather than just the training subset before testing against the validation sets.

3.2.2 Partial replication of Dhahri et al.

To benchmark the real-world performance of the simple Genetic Algorithm (GA) in meta-learning classifiers, this work attempts to partially replicate the work by Dhahri et al. [5] as a case study. Dhahri et al. analyzed the performance of a similar Genetic Program to build Classification workflows for the Wisconsin Breast Cancer Diagnostic dataset, which consists of 569 individuals, comprising 212 malignant and 357 benign cases. The identities of the individuals are kept anonymous to its users via ID numbers.

This replication was performed over a 60-20-20 spit of the dataset into 3 subsets: training, testing, and validation. The GA was run using the training and testing sets To evaluate the fitness of each individual, the corresponding classification model was built using the individual's specified hyperparameters. The model is then trained and tested on these two subsets respectively and fitness is calculated using the testing performance. The GA was run using 100 trials, each seeded with a different random seed, for 100 generations with a population size of 100. This experiment was not optimized for time and was run using the MSU HPCC in parallel on 25 independent cores in the same cluster. Each core took between 4.5 to 5 hours, totaling at most 125 hours.

This method differs from Dhahri et al. 2019 on four counts:

- **Different parameter space** Dhahri et al. only used the number of kernels as the hyperparameter in the parameter space. This difference was challenging to reproduce because the range of possible values for this hyperparameter was not explicitly stated in the paper and because it was difficult to identify what this meant for some of the chosen algorithms. This work uses the same algorithms in the search space; however, the hyperparameter space in this work accounts for more than one hyperparameter. The search space is explicitly defined in Table 3 and the hyperparameters used for each algorithm is listed in Table 2.
- **Simpler fitness function** Dhahri et al. applied a 10-Fold Cross Validation over the entire dataset to evaluate individual fitness. Using a simpler fitness function is important as this work is being developed for software that is intended to run in close to real time. Exploring alternative fitness functions is an area of future work.
- **Final verification step** This final verification step is important in order to validate the solutions found by the GA in an effort to avoid over-fitting the data.
- **Simple Classification workflow** The work presented in Dhahri et al. includes the addition of two data transformation stages (preprocessing and feature selection). Although this is an area of future work, the results presented in this work do not include these steps. The only

extra step applied is the standard scalar function to preprocess the dataset as it is a typical recommended step.

4 Results

4.1 Scikit-learn tutorial

Figure 1 plots the population mean and the mean of the top 10 solutions found by generation number averaged over the 100 runs of the GA. The shaded regions are two standard deviations from the averages. This shows that the GA was able to rapidly converge onto the best models specified in the tutorial.

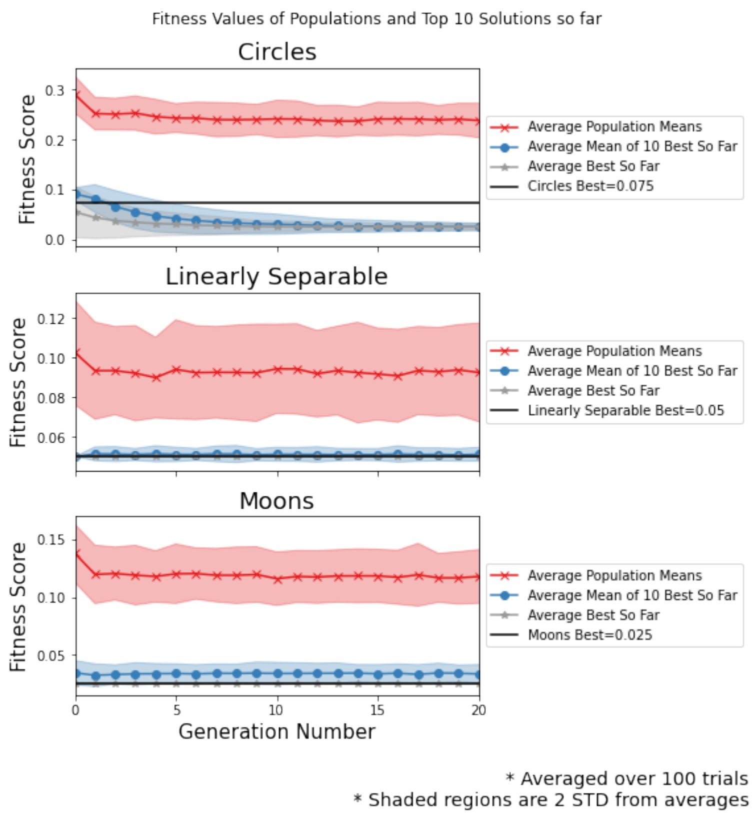

Figure 1: Change in fitness over generations for demo datasets

Table 5 records the data of the top 10 solutions found at the end of the GA over 100 runs for each data set. It shows that the mean fitness score of the top 10 solutions over all 100 trials found at the end of the GA is the same or better than the fitness scores found by the tutorial. The Circles dataset stands out in that by the end of the GA, the top 10 solutions have the same fitness scores.

Table 5: Comparison between GA and tutorial/paper reported accuracy

	Tutorial/paper best accuracy	GA mean	GA std
Circles (Tutorial)	0.075	0.0250	0.0000
Linearly separable (Tutorial)	0.050	0.050	0.0051
Moons (Tutorial)	0.025	0.0250	0.0134
WBCD	0.0176	0.0190	0.0052

4.2 Replicating Dhahri et al.

Figure 2 shows the average mean fitness values of the top 10 solutions found by each generation over all 100 trials. The error regions are plotted with 1 standard deviation. This figure demonstrates that the GA presented in this work is able to converge at a good solution, both in terms of its top 10 best found solutions and that the fitness values of those top 10 are comparable to the best fitness function reported by Dhahri 2019.

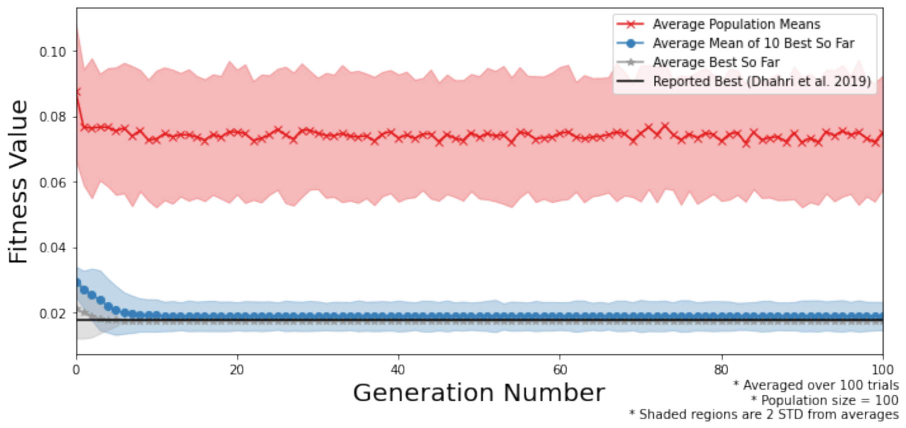

Figure 2: Change in fitness over generations for WBCD dataset

4.3 Validation stage

Figures 3 and 4 show the average performance of each model after training a different portion size of the training set. They suggests that although solutions perform well during the GA search, the final evaluated fitness tends to be greater than the reported fitness during the GA. In addition, these figures suggests that as the amount of training data increases, the performance of the classifiers do not improve greatly. For example, in Figure 4, there is no significant improvement in mean accuracy after a training size of 50, suggesting that the models are not generalizing well. This indicates that either that the data used in the GA is not enough and/or a more complex fitness function is needed.

The default hyperparameters provided with the scikit-learn algorithms for the chosen classifiers are listed in the table below. It shows the best fitness value (0.0176) can already be closely matched using these default parameters. This is interesting in that the default parameters are already the best that our GA can find. It suggests that the default values are good hyperparameters to start with and more interesting datasets need to be explored

5 Concluding discussion

This paper presents the new prototyped SEE-Classify tool which uses a simple GA to search the hyperparamater space of multiple different classification algorithms to find the best parameters for particular problems. The tool was tested using the scikit-learn dataset and the WBCD dataset. Results show that the algorithm quickly converges to solutions that reproduce a prior work. However, one limitation is that the algorithm has not been optimized for time. As such those with access to high compute power can better and more fully utilize this tool over complex problems. Another limitation to this prototype is that a simple fitness function is used. This introduces a bias in favor of algorithms that can more correctly classify majority subgroups found in the provided data. This present a potential negative societal impact if this tool is used to analyze and directly comment on societally relevant data. While this tool was benchmarked on a real-world dataset (WBCD), we are not and do not intend to make any suggestions with regards to the field of medical science. Future work includes testing the library on more complex problems, including a wider options of different fitness functions, and optimizing the Genetic Algorithm before presenting this prototype as a more completed and mature tool. We will also integrate documentation and tutorials inside of the tool in an effort to build a scaffolded-learning tool that can help researchers new to classification, explore and learn about

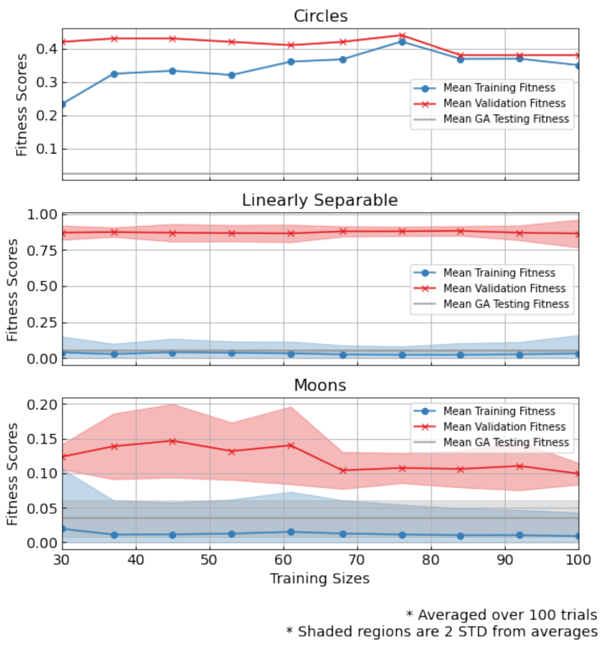

Figure 3: Learning curve of GA best solutions for demo datasets

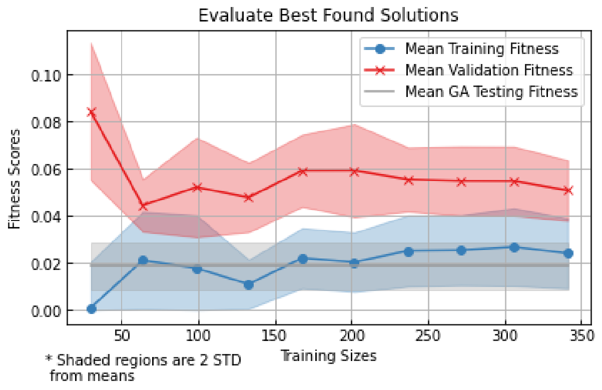

Figure 4: Learning curve of GA best solutions for WBCD

the different algorithms that are available. All of the SEE-Classify software has been released and incorporated into the SEE-Segment project [3], which is provided under an open source license on GitHub, and researchers are encouraged to try the software and contribute to the project.

References

[1] Scikit-learn classifier comparison demo example. https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html. Accessed: 2021-09-01.

[2] See-insight, . URL https://see-insight.github.io/. Accessed: 2021-09-11.

[3] See-segment, . URL https://github.com/see-insight/see-segment. Accessed: 2021-09-11.

[4] M. Claesen and B. De Moor. Hyperparameter Search in Machine Learning. *arXiv:1502.02127 [cs, stat]*, Apr. 2015. URL http://arxiv.org/abs/1502.02127. arXiv: 1502.02127.

[5] H. Dhahri, E. Al Maghayreh, A. Mahmood, W. Elkilani, and M. Faisal Nagi. Automated Breast Cancer Diagnosis Based on Machine Learning Algorithms. *Journal of Healthcare Engineering*, 2019:4253641, 2019. ISSN 2040-2309. doi: 10.1155/2019/4253641.

[6] D. Dua and C. Graff. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

[7] L. A. Ferreira, F. G. Guimaraes, and R. Silva. Applying Genetic Programming to Improve Interpretability in Machine Learning Models. In *2020 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8, Glasgow, United Kingdom, July 2020. IEEE. ISBN 978-1-72816-929-3. doi: 10.1109/CEC48606.2020.9185620. URL https://ieeexplore.ieee.org/document/9185620/.

[8] T. Hospedales, A. Antoniou, P. Micaelli, and A. Storkey. Meta-Learning in Neural Networks: A Survey. *arXiv:2004.05439 [cs, stat]*, Nov. 2020. URL http://arxiv.org/abs/2004.05439. arXiv: 2004.05439.

[9] C. Li, J. Jiang, Y. Zhao, R. Li, E. Wang, X. Zhang, and K. Zhao. Genetic Algorithm based hyper-parameters optimization for transfer Convolutional Neural Network. page 20.

[10] 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*, 12:2825–2830, 2011.

[11] J. R. Rice. The Algorithm Selection Problem**This work was partially supported by the National Science Foundation through Grant GP-32940X. This chapter was presented as the George E. Forsythe Memorial Lecture at the Computer Science Conference, February 19, 1975, Washington, D. C. In M. Rubinoff and M. C. Yovits, editors, *Advances in Computers*, volume 15, pages 65–118. Elsevier, Jan. 1976. doi: 10.1016/S0065-2458(08)60520-3. URL https://www.sciencedirect.com/science/article/pii/S0065245808605203.

[12] K. A. Smith-Miles. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys*, 41(1):6:1–6:25, Jan. 2009. ISSN 0360-0300. doi: 10.1145/1456650.1456656. URL https://doi.org/10.1145/1456650.1456656.

[13] A. S. Wicaksono and A. Afif. Hyper Parameter Optimization using Genetic Algorithm on Machine Learning Methods for Online News Popularity Prediction. *International Journal of Advanced Computer Science and Applications*, 9(12), 2018. ISSN 21565570, 2158107X. doi: 10.14569/IJACSA.2018.091238. URL http://thesai.org/Publications/ViewPaper?Volume=9&Issue=12&Code=ijacsa&SerialNo=38.

[14] W. Wolberg, W. Street, and O. Mangasarian. Breast Cancer Wisconsin (Diagnostic). UCI Machine Learning Repository, 1995.

[15] X. Xiao, M. Yan, S. Basodi, C. Ji, and Y. Pan. Efficient Hyperparameter Optimization in Deep Learning Using a Variable Length Genetic Algorithm. *arXiv:2006.12703 [cs]*, June 2020. URL http://arxiv.org/abs/2006.12703. arXiv: 2006.12703.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] The main claim made in the abstract that this work released and benchmarked of a prototype of a software system that uses a Genetic Algorithm to search an algorithm space of classification. This is demonstrated in section 4 where the performance of the software is discussed and section 5 where we provide a link to the project as reference [3].
   (b) Did you describe the limitations of your work? [Yes] In section 4.3 we discuss the weakness (low validation performance) of this work and in section 5 we discuss future works. We also discuss limitations in the Concluding discussion sction.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] We discuss potential negative societal impacts of this work as consequences of the limitations of this work in the Concluding discussion section.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes] We have ensured and believe that our paper conforms to the ethics review guidelines. While this research uses human-derived data, we mention that the identities of the individuals have been anonymized. In addition, the first half experimentation performed in this work relies on demo data (i.e. moons, circles, linearly separable data) that is generated mathematically and not derived from humans. Since this work is intended to present and discuss the limitations of a first prototype of a research support tool, we do not believe that it crosses any major or systemic ethical concerns. However, we acknowledge potential ethical concerns in the Concluding discussion section.
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] For reproducibility, the link to the Juptyer notebook source codes that were used to generate the figures in this work are included in the appendix. The instructions to reproduce results can be found within the Jupyter notebooks themselves.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Tables 2, 4 for hyperparameters. See sections 3.2.1 and 3.2.2 for data splits.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See the captions on the lower right corners of Figures 1, 2, 3 and on the lower left corner of Figure 4.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See sections 3.2.1 and 3.2.2.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] We cited the existing assets and creators via references [3], [6], [10]. [14].
   (b) Did you mention the license of the assets? [Yes] See section 1 for licenses of the SEE-Insight tools [2] (which includes SEE-Segment [3]) and Scikit-learn package [10], and section 3.1 for the license of the Wisconsin Breast Cancer Diagnostic Dataset [14].
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] We provided the URL to the existing SEE-Segment project (see reference [3]) because rather than releasing new assets under its own name, this work was directly incorporated to that existing project.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [No] The original source of the published dataset and the relevant papers that we looked at did not mention how consent was obtained from people. Because of this limitation, we were unable to discuss how consent was originally obtained.

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [Yes] We mention that the datasets utilized here had been anonymized and therefore are not personally identifiable. See section 3.2.2.

5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Appendix

The source code that was used to generate the figures in this work can be found in the Juptyer Notebooks at `https://github.com/see-insight/see-segment/tree/master/see_classify_figures`.