Tuning Support Vector Machines and Boosted Trees Using Optimization Algorithms

Jill F. Lundell[1],*

[1]*Department of Data Science, Dana-Farber Cancer Institute, Department of Biostatistics, Harvard T.H. Chan School of Public Health, Boston, MA, USA*

Abstract

Statistical learning methods have been growing in popularity in recent years. Many of these procedures have parameters that must be tuned for models to perform well. Research has been extensive in neural networks, but not for many other learning methods. We looked at the behavior of tuning parameters for support vector machines, gradient boosting machines, and adaboost in both a classification and regression setting. We used grid search to identify ranges of tuning parameters where good models can be found across many different datasets. We then explored different optimization algorithms to select a model across the tuning parameter space. Models selected by the optimization algorithm were compared to the best models obtained through grid search to select well performing algorithms. This information was used to create an R package, `EZtune`, that automatically tunes support vector machines and boosted trees.

Keywords *machine learning; optimization; R programming*

1 Introduction

Statistical learning models have gained in popularity in recent years because of their ability to provide greater predictive accuracy than traditional statistical models in many situations. Some models, such as Random Forests, perform well without parameter tuning, but most learning methods have parameters that must be tuned (Breiman, 2001). The No Free Lunch theorems state that there is no one type of model that outperforms all other models in all situations (Schumacher et al., 2001). Thus, it is essential to explore several different types of models to find an optimal solution to a problem. Support vector machines (SVMs) (Cortes and Vapnik, 1995), gradient boosting machines (GBMs) (Friedman, 2001), and adaboost (Freund and Schapire, 1997) are three supervised learning models that can be very accurate when tuned. However, parameters can be difficult to tune and recommendations for tuning methods in journal articles, blogs, and FAQs are often not well justified. Better understanding of the properties of tuning parameters and how to tune them is needed to effectively use these models. Software tools that allow users to tune models without requiring the user to do substantial research are lacking. Further development of tuning software would provide many data analysts with a wide range of more accessible tools for modeling.

We explore tuning parameters for SVM, GBM, and adaboost to identify tuning parameter spaces that yield accurate predictive models. We then search over the tuning parameter space using different optimization algorithms to assess the ability of each algorithm to find a well tuned

---

model over many datasets. This information was used to create an R package called `EZtune` that autotunes SVMs, GBMs, and adaboost. `EZtune` has a simple user interface, is easy to use, has fast computational time, and produces well tuned models. `EZtune` is available on CRAN and specifics about the package and its performance are in Lundell (2019) and Lundell (2023).

2 Overview of Tuning Parameters

The following overview of SVMs and boosted trees briefly summarizes SVMs, GBMs, and adaboost and identifies the tuning parameters for each model.

2.1 Support Vector Machines

SVMs use separating hyperplanes to create decision boundaries for classification and regression models (Cortes and Vapnik, 1995). The separating hyperplane is called a soft margin because it allows some points to be on the wrong side of the hyperplane. The cost parameter, C, dictates the tolerance for points to be on the wrong side of the margin. A large value of C allows many points to be on the wrong side while smaller values of C have a much lower tolerance for misclassified points. A kernel, K, is used to map the classifier into a higher dimensional space. Hyperplanes are used to classify in the higher dimensional space, which results in non-linear boundaries in the original space. The SVM is modeled as:

$$f(x) = \beta_0 + \sum_{i \in S} \alpha_i K(x, x_i; \gamma) \tag{1}$$

where, K is a kernel with tuning parameter γ, S is the set of support vectors (points on the boundary of the margin), and α_i computed using C and the margin. The tuning parameters for SVM classification are C and γ. Common kernels are polynomial, radial, and linear. We use the radial kernel for this work because it performs well in many situations.

Support vector regression (SVR) has an additional tuning parameter, ϵ. SVR attempts to find a function, or hyperplane, such that the deviations between the hyperplane and the responses, y_i, are less than ϵ for each observation (Smola and Schölkopf, 2004). The cost represents the number of points that can be further than ϵ away from the hyperplane. Essentially, SVMs maximize the number of points that are on the correct side of the margin and SVR maximizes the number of points that fall within ϵ of the margin. The only mathematical restriction for the tuning parameters for SVM and SVR is that they are greater than 0.

2.2 Boosted Trees

Boosted trees are part of the family of ensemble methods which combine weak classifiers into a single, accurate, classifier. A weak learner typically does not perform well alone, but combining many weak learners can create a strong classifier (Hastie et al., 2009). With boosted trees, a small tree computed from the training data is the weak learner. The misclassified points or residuals from the tree are examined and the information is used to fit a new tree. The model is updated by adding the new tree to the previously fitted trees. The ensemble is iteratively updated in this manner and final predictions are made by a weighted vote of the weak learners.

The primary difference between boosted tree algorithms is the method used to learn from misclassified observations at each iteration. Adaboost fits a small tree to the training data while applying the same weight to all observations in the dataset (Freund and Schapire, 1997). The

misclassified points are then given greater weight than the correctly classified points and a new tree is computed. The new tree is added to the previous tree with weights. The process is repeated many times with misclassified points being given greater weight and a new tree is created using the weighted data and added to the previous model. Weak learners that perform better are given greater weight than those that classify more poorly. This results in an additive model where the final predictions are the weighted sum of the predictions made by all of the models in the ensemble (Hastie et al., 2009).

GBMs are a boosted tree that uses gradient descent to minimize a loss function during the learning process (Friedman, 2001). The loss function can be tailored to the problem being solved. The MSE was used as the loss function for regression models and a logarithmic loss was used for classification problems in this article. A decision tree is used as the weak learner and trees are kept small to ensure that they are weak. GBMs recursively fit new trees to the residuals from previous trees and then combine the predictions from all of the trees to obtain a final prediction.

Adaboost and GBMs have a nearly identical set of tuning parameters. Both models require that the number of iterations, depth of the trees, and the shrinkage, which controls how fast the trees learn, are tuned. GBMs have an additional tuning parameter which is the minimum number of observations in the terminal nodes.

3   Methods

Tuning parameters were analyzed using six datasets with a binary response and seven with a continuous response. Table 1 lists the datasets and their characteristics. An extensive grid search was done for each dataset to understand the response surface of the mean squared error (MSE) or classification accuracy across tuning parameter values. The response surface was used to identify a tuning parameter space that contained several of the best fitting models for each of the data sets. Optimization algorithms searched through the tuning parameter spaces to find a well tuned model. The MSEs or classification accuracies of the optimization algorithms were compared to the best fitting model found in the grid search to determine which optimization algorithms performed well both in general and relative to each other. The error measure and computation time were both considered when assessing the performance of each optimization algorithm. Computation time was included as a performance measure to ensure that implementation of selected algorithms is practical.

4   Tuning Parameter Spaces

We searched blog posts, books, and journal articles to determine the tuning parameter ranges that have been used by other authors. Many authors do not address all of the tuning parameters and parameter space selection was not backed up by computation in any resources that we found. We used all of the tuning parameters identified in Section 2 and did a grid search over a wide range of tuning parameter values. These ranges include all of the regions seen in other resources and are extended even further for some tuning parameters. Tables 2 and 3 show the ranges that were used for the grid search along with the tuning parameter spaces we recommend. The results of the grid search indicated that the grids were sufficiently large and do not need to be expanded beyond the limits shown in Tables 2 and 3.

SVM, GBM, and adaboost models were computed for tuning parameters throughout their specified grid region. The error measure was evaluated at each grid location using 10-fold cross-

Table 1: List of datasets used to explore tuning parameters.

Dataset	Number of Observations	Number of Dataset	Number of Categorical Explanatory Variables	Number of Continuous Explanatory Variables
Binary				
Breast Cancer Data[1]	699	10	0	9
Ionosphere[1]	351	35	1	33
Pima Indians[1]	768	9	0	8
Sonar[1]	208	61	0	60
Lichen[2]	840	40	2	31
Mullein[2]	12094	32	0	31
Continuous				
Abalone[3]	4177	9	1	7
Boston Housing 2[1]	506	19	1	15
CO2[4]	84	5	3	1
Crime[5]	47	14	1	12
Ames Housing[6]	1460	61	36	24
Union[5]	50	4	0	3
Wage[5]	39	10	0	9

[1]mlbench (Newman et al., 1998); [2]EZtune (Lundell, 2017); [3]AppliedPredictiveModeling (Kuhn and Johnson, 2018); [4]datasets (R Core Team, 2022); [5]Practicing Statistics (Kuiper and Sklar, 2013); [6]Kaggle (De Cock, 2011; Kaggle, 2019)

Table 2: Tuning parameter ranges for models with a binary response.

Model	Parameter	Grid Search Range	Tuning Parameter Space	Search Start Location
Support vector machines	Cost	$[2^{-10}, 2^{25}]$	$[1, 1024]$	10
	γ	$[2^{-25}, 2^{10}]$	$[2^{-10}, 2^{10}]$	2^{-5}
Gradient boosting machines	Number of trees	$[50, 20{,}000]$	$[50, 3000]$	500
	Interaction depth	$[1, 19]$	$[1, 15]$	5
	Shrinkage	$[0.001, 0.1]$	$[0.001, 0.1]$	0.1
	Minimum number of observations in terminal nodes	$[5, 15]$	$[5, 12]$	8
Adaboost	Number of trees	$[100, 1400]$	$[50, 500]$	300
	Interaction depth	$[1, 20]$	$[1, 10]$	10
	Shrinkage (ν)	$[0.01, 1]$	$[0.01, 0.5]$	0.05

validation for most datasets and 3-fold cross-validation for the largest datasets. The error measure is the misclassification rate for binary response variables and the MSE for continuous

Table 3: Tuning parameter ranges for models with a continuous response.

Model	Parameter	Grid Search Range	Tuning Parameter Space	Search Start Location
Support vector machines	Cost	$[2^{-10}, 2^{25}]$	$[1, 1024]$	2
	γ	$[2^{-25}, 2^{10}]$	$[2^{-10}, 2^{0}]$	2^{-5}
	ϵ	$[0, 10]$	$[0, 0.5]$	0.4
Gradient boosting machines	Number of trees	$[50, 20{,}000]$	$[50, 5000]$	2000
	Interaction depth	$[1, 19]$	$[1, 15]$	8
	Shrinkage	$[0.001, 0.1]$	$[0.001, 0.1]$	0.1
	Minimum number of observations in terminal nodes	$[5, 15]$	$[5, 10]$	5

response variables. Computation time was recorded for each of the models to aid in finding regions that are computationally practical. We were able to identify parameter spaces for each model type that included accurate models and faster computation times for each dataset. The parameter space for each model type is identified in Section 4 and specified in Tables 2 and 3. Calculations were done using the packages `e1071` (Meyer et al., 2022), `gbm` (Greenwell et al., 2022), and `ada` (Culp et al., 2016) in the R statistical software environment for statistical computing and graphics (R Core Team, 2022). Note that regression models were not computed with adaboost because the adaboost package that was used does not do regression.

We wanted to ensure that selected tuning parameter spaces produced models with stable performance. Thus, error measures obtained from each fold of cross-validation were used to compute a 95% upper confidence limit for the error measures to assess the consistency of error measures across the grid. The upper confidence limits were graphed at each grid location in addition to the cross validated error measure and the computation time. In all cases, the surfaces produced by the upper confidence limits yielded nearly identical surface patterns as the error measures so they are not discussed further.

Grid searches were done using the data listed in Table 1 over the parameter ranges specified in Tables 2 and 3. Figure 1 shows the surface of the errors obtained for the classification SVM models. Although a distinct surface emerges across all of the datasets, it is difficult to determine a smaller region where performance is good across all datasets. The grid results were subsetted to include only best 20% of the errors and to include the best 20 error measures across the entire grid. Figure 2 shows the surface for the computation times across the grid with the fastest 20 computation times highlighted in orange. Surface plots for the other models are included in the supplementary materials.

The wide distribution of the orange dots in Figures 1 and 2 shows that there are many local minima across the surface. Figure 2 also shows that there are areas in the grid with slow computation times. The best computation times seemed to be in the same grid regions with the best error rates. The MSE and computation time surfaces for the regression models were similar to those for the classification models. Smaller values of ϵ produced smaller MSEs but also had slower computation times for all datasets. Good error rates with reasonable computation times can be obtained by models with a cost between 1 and 1000 and a γ between 2^{-10} to 2^{10}. The
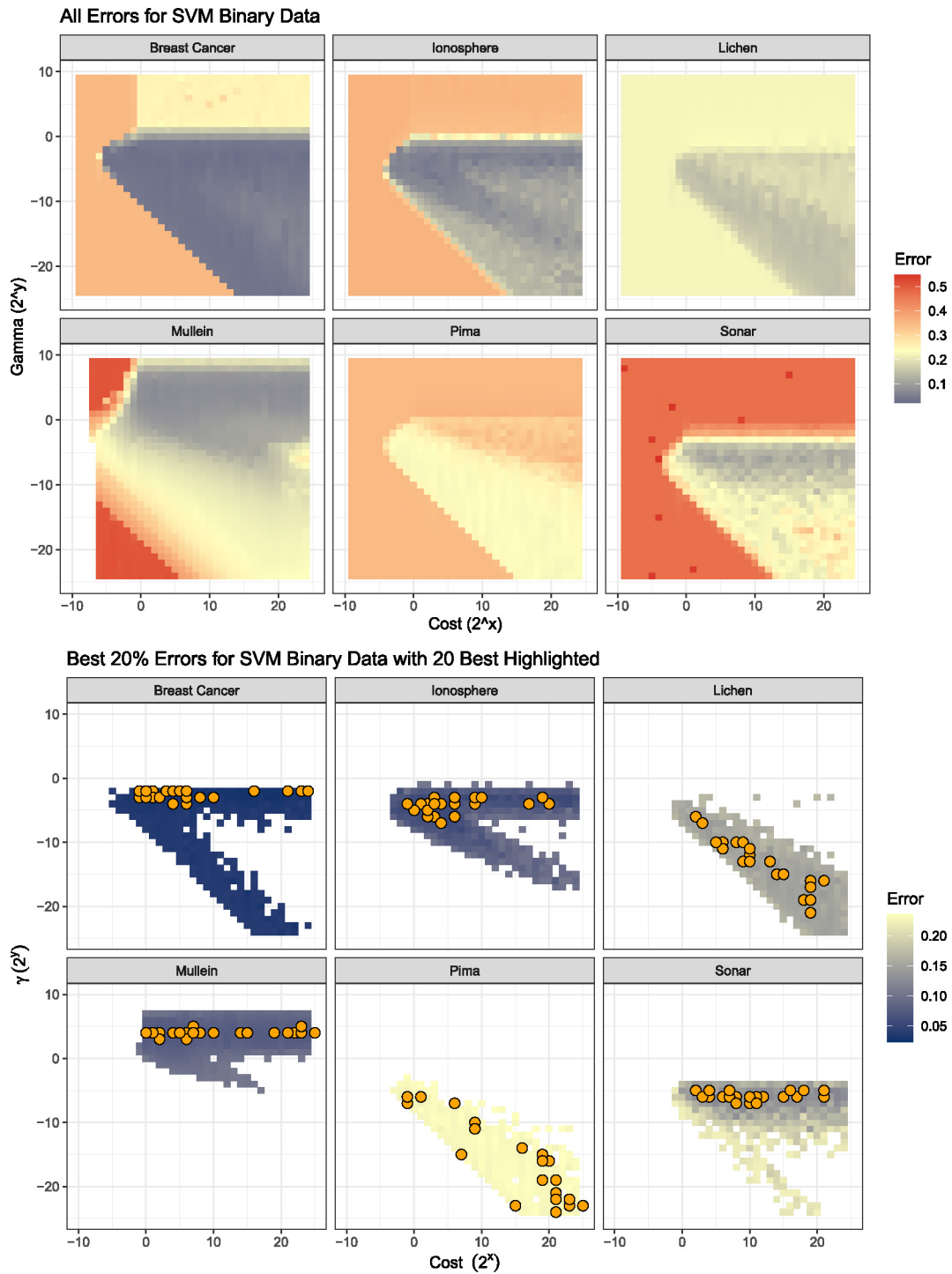
Figure 1: Error surface plots for support vector machines on datasets with a binary response. Orange dots on the bottom figure represent the best 20 models across the grid.

best results for regression were seen for values of ϵ less than 0.5. It is clear from the analysis that cost, γ, and ϵ should all be tuned. Although the regression and classification models had similar
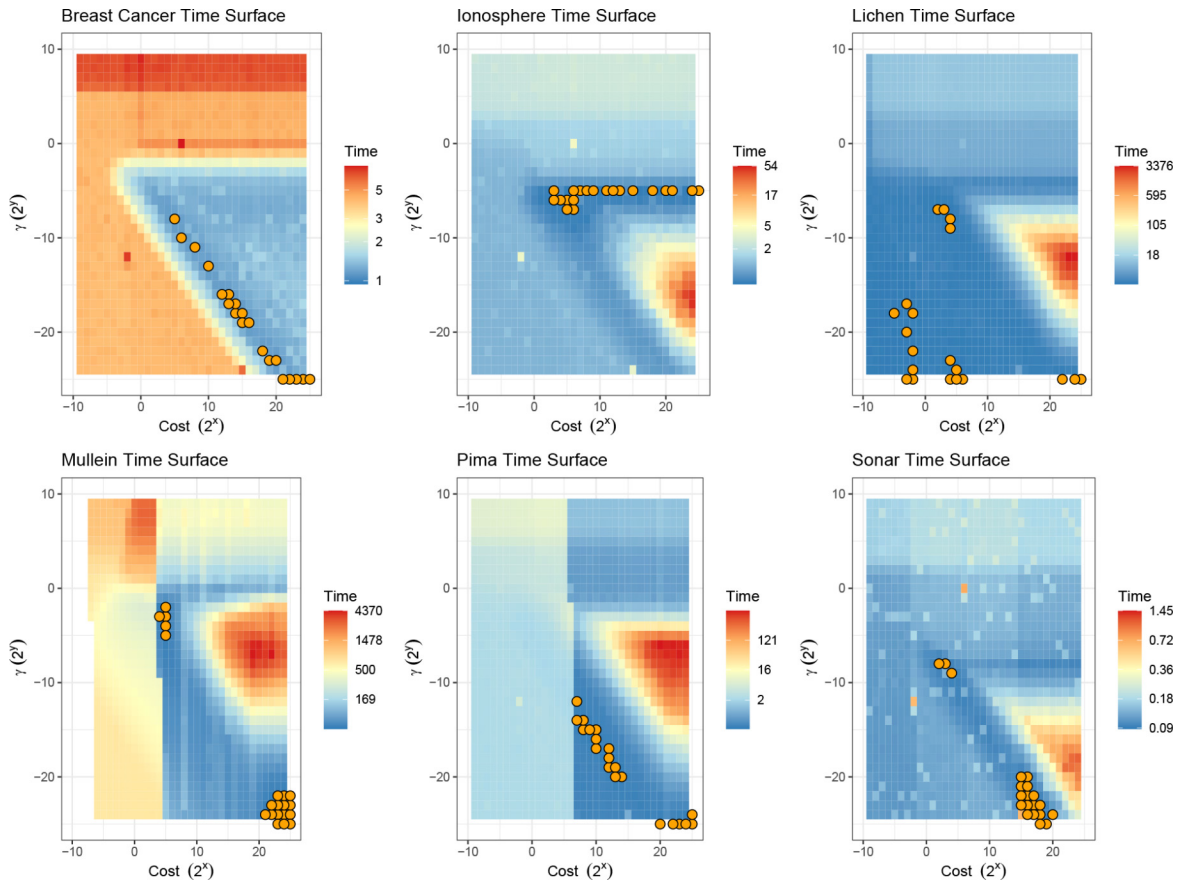
Figure 2: Computation time surface plots for support vector machines on datasets with a binary response. Orange dots represent the 20 models with the shortest computation times across the grid. Time is in seconds.

response surfaces, the tuning parameter spaces selected for data types are slightly different. This is so the subtle differences between each model type can be best utilized. Tables 2 and 3 show the selected tuning parameter spaces for all of the models. Starting values for each of the parameter spaces were also selected from the error surfaces. The starting locations were selected from areas that tend to have low error measures and faster computation times across all datasets.

The parameter regions for GBM were searched in a similar manner and response surfaces showed patterns in performance. However, none of the error rate or computation time surfaces were smooth, even when examined with multidimensional graphics. Computation times were unilaterally faster with smaller values for all tuning parameters with the exception of shrinkage. The best error measures were found across the range of shrinkage values, so a smaller shrinkage does not always result in a better model fit. Better error rates were also found with fewer than 1000 iterations and often with only about 500 iterations. Good results were seen across the spectrum of tested interaction depths and the different values of the minimum number of observations in the terminal nodes. The areas of best performance varied for each dataset so it was determined that the range of values should be minimally trimmed for those two tuning parameters. As with the SVM analysis, it was clear that it is important to tune all tuning parameters. Tables 2 and 3 show the selected tuning parameter spaces.

Adaboost was evaluated only for the binary datasets. The best computation times were seen for the smallest number of trees and the smallest tree depths. Shrinkage did not have much impact on computation times. Good error rates were seen across all values of shrinkage that were tested and good models were found for all values of tree depth and the number of iterations. The tuning parameter space was chosen to try to minimize computation time while catching some of the best models for each dataset. Tables 2 and 3 shows the selected tuning parameter spaces.

Smaller regions than those listed in Tables 2 and 3 were tested during the optimization phase to determine if reducing this region to a smaller area improves computation time with little sacrifice in accuracy. It was found that smaller regions did not decrease computation times for most optimization algorithms and often resulted in larger error measures so the larger parameter space was retained.

5 Optimization Algorithms

Once a parameter space was determined, the parameter spaces for each dataset were searched using 17 different optimization algorithms. Table 4 lists the algorithms that were assessed. Ten searches were carried out for each dataset and algorithm. Both the error measure and computation time were computed for each search to determine the stability of the algorithm. If an algorithm was not able to complete 10 runs within a specified time frame for one of the datasets, it was considered a failure for that dataset.

The R statistical package (R Core Team, 2022) was used for all optimization computations. Table 4 shows the R packages and functions that were used for the optimization. Computation time and error measures were compared and it was assumed that different optimization algorithms may perform better for different model types. A description of each of these algorithms can be found in Appendix A of the supplementary material.

Table 5 concisely summarizes the results of all of the optimization algorithms. Figures 3–7 show parallel coordinate plots of the results of the optimization tests. Values in the graphs have been standardized by subtracting the minimum measure obtained from the grid search and then dividing by the maximum resulting value. This means that the worst error measure or computation time for each dataset is 1 and the best error measure or computation time is 0. The x-axis lists the R function name to avoid confusion for algorithms that were tested with more than one function. If the computation time was too long to obtain a result, a value of 1 was plotted for both the time and the error measure.

6 Discussion

The grid search showed that there are tuning parameter spaces that contain models with small error rates and fast computation times across all of the tested datasets. Areas that have fast computation times for the SVM models include models with good error rates for binary classification. Regression models with SVM showed that this was true for cost and γ, but not always for ϵ. Tuning parameter spaces for binary and continuous responses were similar, but a smaller region for γ can be used when the response is continuous.

GBM and adaboost have nearly identical tuning parameters, but they behave differently. GBM requires a larger range of trees and interaction depths than adaboost. GBM also requires smaller shrinkage values than adaboost. With both models, a smaller shrinkage was not always better but also did not seem to increase computation time.

Table 4: Optimization algorithms along with the packages and functions in R that will be used to implement them.

Algorithm	Type	Package	Function
Ant Lion (Mirjalili, 2015a)	Metaheuristic	MetaheuristicOpt (Septem Riza et al., 2017)	ALO
BOBYQA (Powell, 2009)	Derivative free	minqa (Bates et al., 2022)	bobyqa
Dragonfly (Mirjalili, 2016a)	Metaheuristic	MetaheuristicOpt	DA
Firefly (Yang, 2009)	Metaheuristic	MetaheuristicOpt	FFA
Genetic algorithm (Goldberg, 1999)	Metaheuristic	GA (Scrucca, 2013)	ga
Grasshopper (Saremi et al., 2017)	Metaheuristic	MetaheuristicOpt	GOA
Grey wolf (Mirjalili et al., 2014)	Metaheuristic	MetaheuristicOpt	GWO
Hooke-Jeeves (Hooke and Jeeves, 1961)	Derivative free	optimx (Nash, 2014a), dfoptim (Varadhan et al., 2020)	hjk, hjkb
Improved harmony search (Mahdavi et al., 2007)	Metaheuristic	MetaheuristicOpt	HS
L-BFGS (Byrd et al., 1995)	Quasi-Newton	lbfgsb3 (Nash et al., 2020), stats (R Core Team, 2022)	lbfgsb3, optim
Moth flame (Mirjalili, 2015b)	Metaheuristic	MetaheuristicOpt	MFO
Nelder-Mead (Kelley, 1999)	Derivative free	dfoptim	nmk
Nonlinear conjugate gradient (Dai and Yuan, 2001)	Gradient	Rcgmin (Nash, 2014b)	Rcgmin
Particle swarm (Shi and Eberhart, 1998)	Metaheuristic	MetaheuristicOpt	PSO
Sine cosine (Mirjalili, 2016b)	Metaheuristic	MetaheuristicOpt	SCA
Spectral projected gradient (Birgin et al., 2000)	Gradient	BB (Varadhan and Gilbert, 2009)	spg
Whale (Mirjalili and Lewis, 2016)	Metaheuristic	MetaheuristicOpt	WOA

Table 5: Performance summary of optimization algorithms.

| Method	Error	Time	Consistency
Genetic algorithm	Good	Slow	Consistent
Hooke and Jeeves	Varies	Varies	Inconsistent
Hooke and Jeeves B	Good	Good	Consistent
L-BFGS	Good	Good	Crashes often
Nocedal-Morales	Poor	Slow	Consistent
Nonlinear conjugate gradient	Poor	Fast	Stays at start
BOBYQA	Poor	Fast	Consistent
L-BFGS B	Poor	Fast	Consistent
Spectral projected gradient	Moderate to Poor	Good to moderate	Inconsistent
Ant lion	Poor	Moderate	Consistent
Dragonfly	Poor	Good	Consistent
Firefly	Poor	Slow	Consistent
Grasshopper	Moderate	Moderate	Inconsistent
Grey wolf	Poor	Moderate	Inconsistent
Harmony search	Poor	Moderate	Inconsistent
Moth flame	Poor	Moderate	Inconsistent
Particle swarm	Poor	Slow	Consistent
Sine cosine	Poor	Moderate	Inconsistent
Whale optimization	Poor	Slow	Consistent
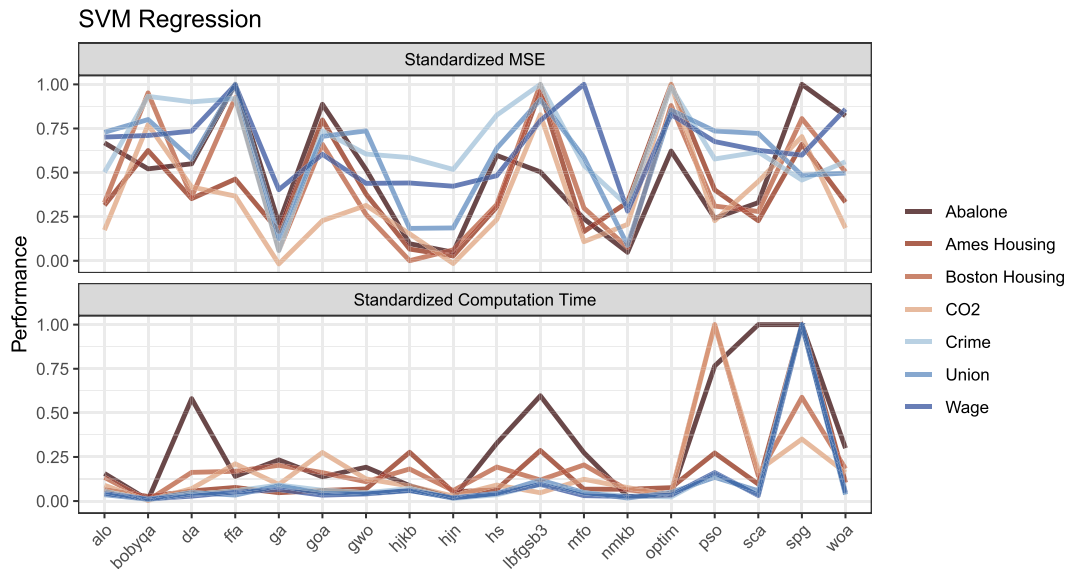
Figure 3: Standardized optimization results for support vector regression.

The optimization algorithms listed in Table 4 differed markedly in computation time and in their ability to find a set of parameters that produce a good model. We initially thought that a gradient method would perform well for the SVM models because the error surfaces were smooth and that non-gradient based algorithms would be better for GBM and adaboost.
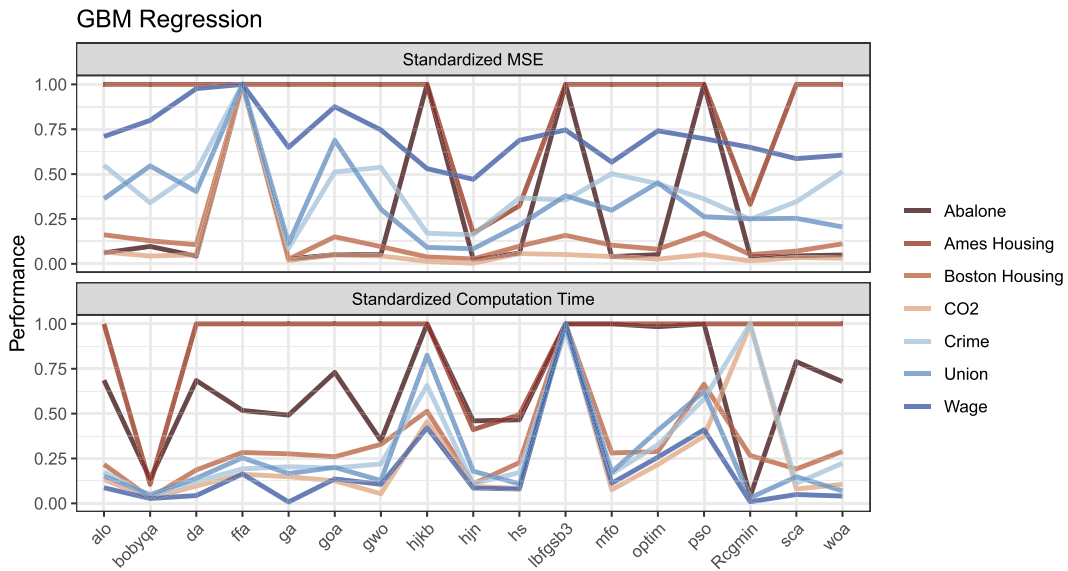
GBM Regression

Figure 4: Standardized optimization results for gradient boosting machine regression.

SVM Binary Classification

Figure 5: Standardized optimization results for support vector machines for binary classification.

However, gradient based methods performed poorly for all models. The non-linear conjugate gradient algorithm had a very fast computation time, but it failed to move from the starting values. This may be an artifact of the `Rcgmin` function that was used (Nash, 2014b), but it is a gradient based function and it is unlikely it will perform well regardless of how it is coded. The Nelder-Meade algorithm had fast computation times and low error rates that rivaled all other algorithms, but it often failed to converge. It is worth exploring this algorithm in another programming language, such as Python, to see if more stable performance can be achieved. The metaheuristic algorithms seem like they would perform well based on the appearance of the error
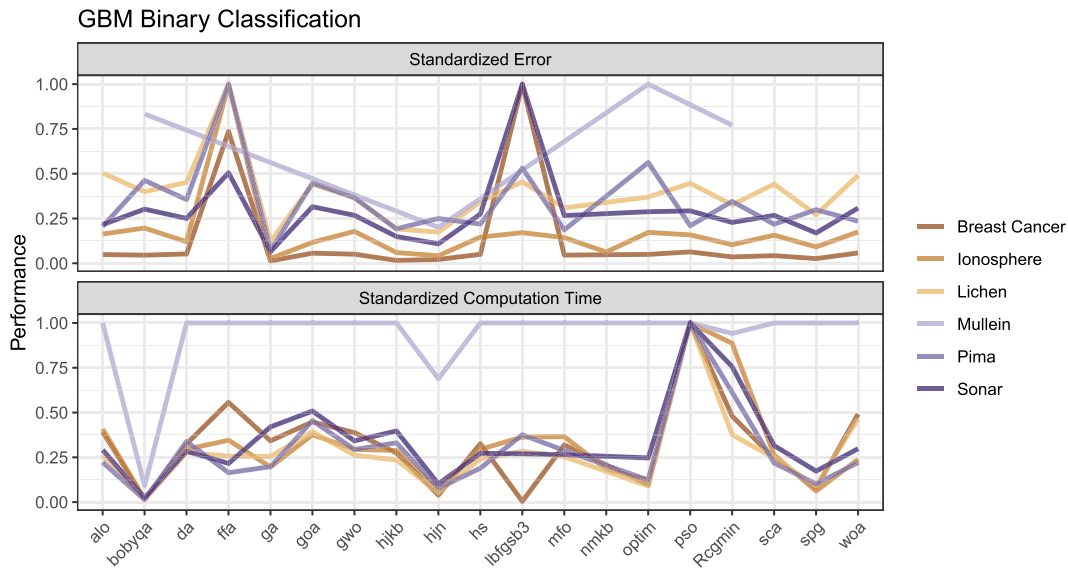
GBM Binary Classification

Figure 6: Standardized optimization results for gradient boosting machines for binary classification.

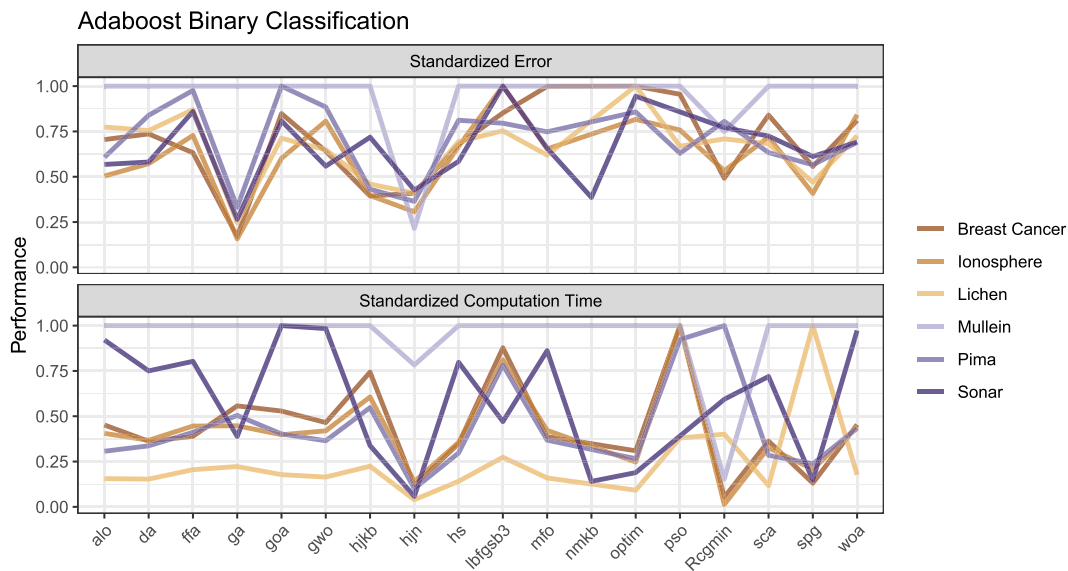Adaboost Binary Classification

Figure 7: Standardized optimization results for adaboost models for binary classification.

rate surfaces and based on the performance of the genetic algorithm so further investigation in Python or another language may yield better results. The Hooke-Jeeves algorithm consistently produced the best error measures and computation times for all datasets across all model types. The genetic algorithm found the best error rates overall, but computation times were slow. With larger datasets, the computation time of the genetic algorithm was prohibitive.

The success in finding tuning parameter spaces across multiple datasets and the identification of two optimization algorithms that were able to find well tuned models makes it possible to automate tuning. We wrote an R package, EZtune (Lundell 2019 & 2023), that can tune an

SVM, GBM, or adaboost model with little input from the user. The user selects a model type in the main function, `eztune`, and the function searches the tuning parameter spaces listed in Tables 2 and 3 using either a Hooke-Jeeves or genetic algorithm. The function returns a tuned model, the values for the tuning parameters, and performance metrics. The results of the `eztune` function were compared to the grid search results and the models that were returned were as good, or nearly as good, as the best model found in the grid search. The benchmarking results are included in Appendix B in the supplementary material. The `eztune` function also contains arguments that can substantially decrease computation time with little sacrifice in model performance. The package contains a second function, `eztune_cv` that will compute a cross-validation accuracy rate or MSE for any model generated using `eztune`.

7   Conclusions

This computational work demonstrates that practical tuning parameter spaces can be identified for SVMs and boosted trees that work well across many different datasets. These spaces can increase the speed and efficacy of tuning an SVM or boosted tree. In addition to narrowing the search region, we have demonstrated that the Hooke-Jeeves algorithm and a genetic algorithm can search the tuning parameter space to identify a tuned model that is often as good, or nearly as good, as a model found by a very extensive grid search. The information in this article not only provides insight into tuning these types of models, but it can also be used to create algorithms that automatically tune SVMs and boosted trees well with reasonable computation time. The R package `EZtune` was created using this research and provides an effective algorithm for tuning SVMs and boosted trees with a user friendly interface.

Supplementary Material

The following supplementary material are available:

Appendixes

A: Description of optimization algorithms
B: Performance tables

R-package for EZtune: R-package EZtune that can implement autotuning of SVMs, GBMs, and adaboost using the Hooke-Jeeves algorithm and genetic algorithm. The package also contains Lichen and Mullein datasets used in the examples in the article. The package is currently available on CRAN and updates are available at https://github.com/jillbo1000/EZtune (GNU zipped tar file).

Code and data for creating grids and performing optimization tests: The code and data used to create the error and time response surfaces and the code for testing the optimization algorithms is available at https://github.com/jillbo1000/autotune.

References

Bates D, Mullen KM, Nash JC, Varadhan R (2022). minqa: Derivative-free optimization algorithms by quadratic approximation. R package version 1.2.5.

Birgin EG, Martínez JM, Raydan M (2000). Nonmonotone spectral projected gradient methods on convex sets. *SIAM Journal on Optimization*, 10(4): 1196–1211. https://doi.org/10.1137/S1052623497330963

Breiman L (2001). Random forests. *Machine Learning*, 45(1): 5–32. https://doi.org/10.1023/A:1010933404324

Byrd RH, Lu P, Nocedal J, Zhu C (1995). A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(5): 1190–1208. https://doi.org/10.1137/0916069

Cortes C, Vapnik V (1995). Support-vector networks. *Machine Learning*, 20(3): 273–297.

Culp M, Johnson K, Michailidis G (2016). ada: The R package ada for stochastic boosting. R package version 2.0-5.

Dai YH, Yuan Y (2001). An efficient hybrid conjugate gradient method for unconstrained optimization. *Annals of Operations Research*, 103(1–4): 33–47. https://doi.org/10.1023/A:1012930416777

De Cock D (2011). Ames, Iowa: Alternative to the Boston housing data as an end of semester regression project. *Journal of Statistics Education*, 19: 3.

Freund Y, Schapire RE (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1): 119–139. https://doi.org/10.1006/jcss.1997.1504

Friedman JH (2001). Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29(5): 1189–1232. https://doi.org/10.1214/aos/1013203451

Goldberg D (1999). *Genetic algorithms in search optimization and machine learning.* Addison-Wesley Longman Publishing Company, Boston, MA, USA.

Greenwell B, Boehmke B, Cunningham J, Developers G (2022). gbm: Generalized Boosted Regression Models. R package version 2.1.8.1.

Hastie T, Tibshirani R, Friedman J (2009). *The elements of statistical learning: Data mining, inference, and prediction.* Springer, New York, NY, USA.

Hooke R, Jeeves TA (1961). "Direct Search" solution of numerical and statistical problems. *Journal of the ACM*, 8(2): 212–229. https://doi.org/10.1145/321062.321069

Kaggle (2019). Ames housing dataset. https://www.kaggle.com/datasets/prevek18/ames-housing-dataset. Accessed: 2019-02-13.

Kelley CT (1999). *Iterative methods for optimization.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

Kuhn M, Johnson K (2018). AppliedPredictiveModeling: Functions and data sets for 'Applied Predictive Modeling'. R package version 1.1-7.

Kuiper S, Sklar J (2013). *Practicing statistics: Guided investigations for the second course.* Pearson, Boston, MA, USA.

Lundell J (2023). Eztune: A package for automated hyperparameter tuning in R. arXiv preprint arXiv:2303.12177.

Lundell JF (2017). There has to be an easier way: A simple alternative for parameter tuning of supervised learning methods. In: *JSM Proceedings, Statistical Computing Section*, 3028–3036. American Statistical Association, Alexandria, VA.

Lundell JF (2019). Tuning hyperparameters in supervised learning models and applications of statistical learning in genome-wide association studies with emphasis on heritability, Ph.D. thesis, Utah State University.

Mahdavi M, Fesanghary M, Damangir E (2007). An improved harmony search algorithm for

solving optimization problems. *Applied Mathematics and Computation*, 188(2): 1567–1579. https://doi.org/10.1016/j.amc.2006.11.033

Meyer D, Dimitriadou E, Hornik K, Weingessel A, Leisch F (2022). e1071: Misc functions of the department of statistics. probability theory group (Formerly: E1071), TU Wien. R package version 1.7-12.

Mirjalili S (2015a). The ant lion optimizer. *Advances in Engineering Software*, 83: 80–98. https://doi.org/10.1016/j.advengsoft.2015.01.010

Mirjalili S (2015b). Moth-flame optimization algorithm: A novel nature-inspired heuristic paradigm. *Knowledge-Based Systems*, 89: 228–249. https://doi.org/10.1016/j.knosys.2015.07.006

Mirjalili S (2016a). Dragonfly algorithm: A new meta-heuristic optimization technique for solving single-objective, discrete, and multi-objective problems. *Neural Computing & Applications*, 27(4): 1053–1073. https://doi.org/10.1007/s00521-015-1920-1

Mirjalili S (2016b). SCA: A sine cosine algorithm for solving optimization problems. *Knowledge-Based Systems*, 96: 120–133. https://doi.org/10.1016/j.knosys.2015.12.022

Mirjalili S, Lewis A (2016). The whale optimization algorithm. *Advances in Engineering Software*, 95: 51–67. https://doi.org/10.1016/j.advengsoft.2016.01.008

Mirjalili S, Mirjalili SM, Lewis A (2014). Grey wolf optimizer. *Advances in Engineering Software*, 69: 46–61. https://doi.org/10.1016/j.advengsoft.2013.12.007

Nash JC (2014a). On best practice optimization methods in R. *Journal of Statistical Software*, 60(2): 1–14. https://doi.org/10.18637/jss.v060.i02

Nash JC (2014b). Rcgmin: Conjugate gradient minimization of nonlinear functions. R package version 2013-2.21.

Nash JC, Zhu C, Byrd R, Nocedal J, Morales JL (2020). lbfgsb3: Limited memory BFGS minimizer with bounds on parameters. R package version 2020-3.2.

Newman D, Hettich S, Blake C, Merz C (1998). UCI repository of machine learning databases. http://www.ics.uci.edu/~mlearn/MLRepository.html

Powell MJD (2009). The BOBYQA algorithm for bound constrained optimization without derivatives. *Cambridge NA Report NA2009/06, University of Cambridge, Cambridge*, 26–46.

R Core Team (2022). *R: A language and environment for statistical computing.* R Foundation for Statistical Computing, Vienna, Austria.

Saremi S, Mirjalili S, Lewis A (2017). Grasshopper optimisation algorithm: Theory and application. *Advances in Engineering Software*, 105: 30–47. https://doi.org/10.1016/j.advengsoft.2017.01.004

Schumacher C, Vose MD, Whitley LD (2001). The no free lunch and problem description length. In: Spector L, Goodman ED, Wu A, Langdon WB, Voight HM (eds.), *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, 565–570. Morgan Kaufmann Publishers Inc.

Scrucca L (2013). GA: A package for genetic algorithms in R. *Journal of Statistical Software*, 53(4): 1–37. https://doi.org/10.18637/jss.v053.i04

Septem Riza L, Iip , Prasetyo Nugroho E (2017). metaheuristicOpt: Metaheuristic for optimization. R package version 1.0.0.

Shi Y, Eberhart R (1998). A modified particle swarm optimizer. In: *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No. 98TH8360)*, 69–73. IEEE.

Smola AJ, Schölkopf B (2004). A tutorial on support vector regression. *Statistics and Computing*,

14(3): 199–222. https://doi.org/10.1023/B:STCO.0000035301.49549.88

Varadhan R, Gilbert P (2009). BB: An R package for solving a large system of nonlinear equations and for optimizing a high-dimensional nonlinear objective function. *Journal of Statistical Software*, 32(4): 1–26. https://doi.org/10.18637/jss.v032.i04

Varadhan R, Hopkins University J, Borchers HW (2020). dfoptim: Derivative-free optimization. In: ABB Corporate Research. R package version 2020.10-1.

Yang XS (2009). Firefly algorithms for multimodal optimization. In: Watanabe O, Zeugmann T (eds.), *International Symposium on Stochastic Algorithms*, 169–178. Springer.