OPTIMIZATION OF A GRADIENT BOOSTING MODEL FOR PREDICTING POTENTIAL FIRE CONTROL LOCATIONS

Technical Report
August 10, 2021

Dennis W. Hallema[1] and Christopher D. O'Connor[2]

[1]North Carolina State University, Department of Forestry and Environmental Resources, Research Triangle Park, NC 27709 (dwhallem@ncsu.edu)

[2]USDA Forest Service Rocky Mountain Research Station, Missoula MT 59801

Abstract

In this study we performed a sensitivity analysis and benchmark calculation of the gradient boosting classifier of PCL 1.0 GBM, a model for predicting potential fire control locations (PCLs) programmed in *R* using the *gbm* package. Data includes 253,513 grid cells of 30 x 30 meters with information on topography (flatdist, valley distance, distance to ridge, cost distance), road network (road distance, barrier distance), rate of spread (ros01), resistance to control (rtc01), and fire suppression difficulty (sdi01), randomly sampled for the 2019 Miller Fire in the Coronado National Forest in southwest Arizona. We created an algorithm to process data outliers, and a wrapper to run the GBM on multiple CPU cores simultaneously. Next, we performed 2,052 simulations on two commercial grade workstations over the course of a week varying 8 parameters: number of trees fitted, interaction depth, minimum number of observations in terminal nodes, learning rate, number of cross-validation folds, cross-validation stratification by class, outlier processing method, and model save option. Our findings suggest that a GBM with 5,000 trees accounting for all possible interactions, can provide as much predictive power in terms of area under precision-recall curve (AUPRC for testing on 30% of data points) as a 20,000-tree GBM but 4 times faster. Large trees did not appear to overfit the training data in terms of AUPRC but provided no additional performance benefit. Likewise, no outlier processing outperformed the three methods for outlier processing (Tukey IQR, interdecile range, and range between the 2.5th and 97.5th percentile).

1 INTRODUCTION

PCL 1.0 GBM (O'Connor *et al.*, 2017) is a development model for predicting potential fire control locations (PCLs), created as part of a research program to develop decision support for pre-fire planning and operational fire management. PCL 1.0 GBM can quantify relationships between topography, fuel characteristics, road networks and fire suppression efforts, and assess their impact on fire perimeter locations. The model then uses the information on these relationships to generate a probability estimate the potential fire for control across a gridded surface.

The core of this empirical machine learning model is formed by an implementation of Friedman's two-class Gradient Boosting Machine (Friedman, 2001) in *R* (R Core Team, 2018), the *gbm* package

(Ridgeway, 2018). This package is frequently used for the analysis of connections between spatial datasets (Hallema *et al.*, 2018), and provides the base algorithms used to compute probability surfaces of environmental or ecological response variables (Leathwick *et al.*, 2008; Elith and Leathwick, 2017; O'Connor *et al.*, 2017).

1.1 Gradient boosting and robustness

Gradient boosting is an iterative model building process, and involves the iterative adding of weak learners (basis functions describing a set of decision trees) to describe a given response variable in a greedy fashion. Greedy implies that the model adds these basis functions in a way that reduces the loss function, without regard for the implications. Gradient boosting follows the gradient of loss, and is in this regard slightly different from AdaBoost (adaptive boosting) where shortcomings are identified by misclassified samples with a high weight (Friedman, 2001). The nature of gradient boosting is such that many trees combined describe progressively more complex aspects of the data and their interactions.

Depending on the type of data provided as response variable, gradient boosting will rely on different assumptions on the statistical distribution of the desired output from the model. In our case, we want the GBM to tell us whether a given sample should be a PCL or not. To describe this type of response we use the Bernouilli distribution, a special case of the binomial distribution. The Bernouilli distribution is a discrete probability distribution that produces a Boolean outcome, viz. PCL or not PCL.

In two-factor (binary) classification problems, the loss function is defined as the negative binomial log-likelihood (Friedman *et al.*, 2000):

$$L(y, F) = \log(1 + \exp(-2F)), y \in \{-1,1\} \quad \text{(Eq. 1)}$$

where

$$F(x) = \frac{1}{2} log \left[\frac{\Pr(y=1|x)}{\Pr(y=-1|x)}\right] \quad \text{(Eq. 2)}$$

It follows from Eq. 1 and Eq. 2 that if the response variable *y* is very large and predictor variable *x* very small for a given iteration, the empirical loss function has almost no dependence on *x*. The *x*,*y* pairs for which this is the case have very little influence on the loss function and are trimmed from the result by setting a weight threshold α (values between 0.05 and 0.2). This is called influence trimming, and is similar to weight trimming in AdaBoost (Friedman, 2001). This strategy improves numerical stability and reduces computer load. In LogitBoost for example, 90-95% of the observations were removed without reduction in prediction accuracy, but with the advantage of reducing computation by a factor 10-20 (Friedman, 2001).

By reducing the influence of extreme values, gradient boosting shares an important advantage with other tree-based methods, namely its robustness. Gradient boosting algorithms are invariant under all strictly monotone transformations (Friedman, 2001). Boosting trees are not sensitive to long-tailed distributions and outliers in input or output, and are robust to adding irrelevant input variables or missing values (Friedman, 2001). The tree-based approach of recursive binary splitting combined with influence trimming capability eliminates the influence of outliers, and removes the need for external imputation schemes (Friedman, 2001), *i.e.* no need for scaling, centering or trimming data.

In the case of PCL 1.0 GBM, one should be able to train on unscaled data and obtain a transferable model. But while gradient boosting offers high accuracy parameterization can be computationally costly. A quantitative sensitivity analysis can help to determine how complex PCL GBMs really need to be (number of trees needed), and whether influence trimming is indeed sufficient, or if outlier processing of the topographic, fuel characteristics, road and fire suppression data improves the PCL GBM performance.

1.2 Aims and approach

The goals of this study were to (1) assess the parameter sensitivity of the PCL 1.0 gradient boosting classifier (O'Connor *et al.*, 2017; hereafter: PCL 1.0, March 1, 2020 version programmed in *R*), (2) measure how fast the model code runs (benchmarking), and (3) formulate recommendations for improving model training time. We defined the following objectives as part of our approach:

1. Write a function for processing outliers in predictor data;
2. Create a PCL 1.0 GBM model wrapper containing the instructions for each simulation, including optional outlier processing;
3. Write a model run script to perform the sensitivity analysis, with support for parallel processing;
4. Define the hyperparameter space we want to explore, and perform a combined sensitivity analysis/benchmarking via random search of the hyperparameter space.

2 METHOD

2.1 Outlier processing function

Outliers are data values that stand out from other values in the same dataset. Outliers are commonly introduced by variability in the physical quantity or process described by the data (these are accurate outliers), observational errors (random or systematic errors associated with observers or instrumentation), or processing of missing values (*e.g.* replacing nodata with a large integer value or zeros). We wrote an outlier processing function that for a given univariate series 1) detects the outliers that fall outside a window defined by upper and lower threshold values, and 2) replaces these outliers with said upper and lower threshold values depending on whether the outlier is a high or low extreme value.

We programmed three methods to process outliers, namely

1. TukeyIQR, to trim values outside the interquartile range plus or minus 1.5 times the interquartile range (Tukey's method),
2. IDR, to trim values outside the interdecile range, *i.e.*, the range between the 10^{th} and 90^{th} percentile (80% of the data samples), and
3. P025975, to trim values outside the range between the 2.5^{th} and 97.5^{th} percentile.

Tukey IQR is the most common method, and does not make any assumption about the existence of outliers. In other words, Tukey IQR only trims outliers when they exist. But IDR and P025975 assume there are always outliers (unless the dataset is so small that the value range matches the quantile range). The best strategy depends on the nature of the outliers, the objective and the sample size. The outlier trimming function was programmed in data.table to minimize computer time.

2.2 Model wrapper

The purpose of the PCL 1.0 GBM model wrapper is to provide the instructions for an individual simulation of the GMB. The wrapper is called for each parameter set we want to test in the sensitivity analysis, and every call initializes the following sequence of tasks:

1. Parse the data and arguments for the outlier trimming function and the gbm() function;
2. Optionally call the outlier processing function;
3. Train the GBM with the parameters provided to it;
4. Optionally save a workspace image;
5. Benchmark tasks 2-4;
6. Create a sensitivity result object;
7. Write a data summary log;
8. Update the sensitivity result log (allow multiple simultaneous connections to corresponding file);
9. Finally, return a result object of class data.frame containing the result object.

2.3 Model run script with support for parallel processing

The sensitivity analysis script is designed to loop the PCL 1.0 GBM model over a selection of parameter value combinations and evaluate model performance at each instance. Specifically, it executes the following sequence of tasks:

1. Set a random seed for sampling;
2. Create and register a "doParallel" computational cluster to enable multithreading;
3. Set file paths and read as data table (fread);
4. Define the GBM hyperparameter grid;
5. Random sampling of the hyperparameter grid;
6. Call the model wrapper (see previous subsection) for each randomly sampled parameter set. Calls are executed in parallel and dispatched to all CPU cores except one. One core is reserved;
7. (Register sequential)

2.4 Sensitivity analysis and benchmarking

Sensitivity analysis and benchmarking was performed on data collected for the Miller Fire that started in the Coronado National Forest (southwest Arizona, 31.449°N 109.027°W) in June 2019 (InciWeb, 2020). This wildfire was sparked by lightning and subsequently burned 23 km^2 of the Bunk Robinson Wilderness Study Area in the Peloncillo Mountains. The data includes a random sample of 253,513 grid cells of 30 x 30 meters and provides information on topography (flatdist, valley distance, distance to ridge, cost distance), road network (road distance, barrier distance), rate of spread (ros01), resistance to control (rtc01), and fire suppression difficulty (sdi01).

Table 1 gives the parameter values that we evaluated in the sensitivity analysis and benchmarking, all other gbm() parameters were kept at their default values. We expanded the parameter values into a parameter grid containing all combinations possible (26,880 sets), and then sampled randomly without replacement. The model wrapper was called for the parameter combinations in the resulting search grid.

Table 1. Parameter values sampled in the sensitivity analysis.

PARAMETER	VALUES						
N.TREES	200	500	1000	2000	5000	10000	20000
INTERACTION.DEPTH	1	2	5	8			
N.MINOBSINNODE	2	5	10				
SHRINKAGE	0.005	0.01	0.02	0.05	0.1		
CV.FOLDS	2	3	5	10			
CLASS.STRATIFY.CV	FALSE	TRUE					
REPLACEOUTLIERS	FALSE	TukeyIQR	IDR	P025975			
SAVEIMAGE	FALSE	TRUE					

By default, gbm() returns the training error (train.error) and the validation error (validation.error). We also used cross-validation mode, gbm() where the samples is folded *k* times (usually between 2 and 10 times) in order to calculate the cross-validation error (cv.error). Finally, we computed model mean-squared error, probabilities, area under the receiver-operator curve (AUROC) and area under the precision-recall curve (AUPROC) for train (30% of samples) and test sets (70% of samples).

How these errors relate to one another is indicative of how efficient the model is and how well it performs. For example, a model with a small training error and large validation error is indicative of overfitting. Given the importance of pinpointing PCLs, we assumed that AUPRC was the lead indictor of model performance, with higher values suggesting better performance. We also fitted two other gradient boosting models, this time not to predict PCLs but to determine the relative influence of individual parameters on AUPRC and benchmark, respectively.

3 RESULTS AND DISCUSSION

3.1 Classification performance

Figure 1 shows how each calibration parameter affects AUPRC and benchmark (elapsed_min), with the best simulations in terms of AUPRC listed in Table 2. In the top ten, we find GBMs larger than 5000 trees and an interaction depth of 8 (full interaction). The best model in terms of AUPRC is a full interaction GBM with 20000 trees (AUPRC=0.4582), however a 5000-tree GBM computed nearly twice as fast with similar performance (AUPRC=0.4523).

AUPRC improves for higher values of interaction.depth and n.trees (Figure 2a and Table 3). The relative influence of the following parameters across the ensemble of simulations (Figure 2 and Table 3) is so small that it falls below the noise level: number of cross-validation folds (cv.folds), cross-validation stratified by class (class.stratify.cv), and minimum number of observations in the terminal nodes of the trees (n.minobsnode). The optional parameter for outlier trimming in data pre-processing (replaceoutliers) has a significant influence (16.56%; see Table 3) on AUPRC, but a negative influence for that: simulations without pre-processing generally outperformed simulations with pre-processing (any of the three methods).

3.2 Benchmarking

Model training time is most affected by the number of trees fitted, followed by the number of cross-validation folds, and the depth of interaction, in order of decreasing effect (Figure 2b and Table 3). Other parameters like shrinkage, outlier replacement, minimum number of observations in terminal nodes, stratification of the cross-validation sample, and model saving option (with compression), have no significant impact on calculation time.

Table 2. Top ten of PCL 1.0 GBM simulations (*n* = 2052) in terms of area under precision-recall curve (AUPRC) for test data (sorted largest to smallest). Parameterization, model performance and benchmark. The outlier trimming algorithms are "TukeyIQR," Tukey's interquartile range plus or minus 1.5 times the IQR, "IDR", the interdecile range, and "P025975", the range between the 2.5th and 97.5th percentile.

RANK (AUPRC TEST)	1	2	3	4	5	6	7	8	9	10
N.TREES	20000	20000	10000	20000	20000	20000	10000	20000	5000	10000
INTERACTION.DEPTH	8	8	8	8	8	8	8	8	8	8
N.MINOBSINNODE	10	5	10	2	5	5	2	2	5	2
SHRINKAGE	0.01	0.01	0.02	0.02	0.01	0.01	0.02	0.02	0.02	0.02
BAG.FRACTION	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
TRAIN.FRACTION	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
CV.FOLDS	2	3	5	2	10	3	5	2	5	5
KEEP.DATA	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
CLASS.STRATIFY.CV	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	TRUE	FALSE	TRUE
N.CORES	1	1	1	1	1	1	1	1	1	1
CLF_LAST_TRAIN.ERROR	0.5999	0.6016	0.6097	0.4984	0.6059	0.6022	0.6022	0.4989	0.6680	0.6016
CLF_LAST_VALID.ERROR	0.7481	0.7511	0.7443	0.7428	0.7508	0.7493	0.7527	0.7447	0.7380	0.7439
CLF_LAST_CV.ERROR	0.7557	0.7484	0.7477	0.7678	0.7458	0.7493	0.7480	0.7707	0.7524	0.7548
MSE_TRAIN	0.0960	0.0962	0.0966	0.0837	0.0967	0.0963	0.0964	0.0838	0.1033	0.0961
AUROC_TRAIN	0.7368	0.7419	0.7452	0.7316	0.7482	0.7428	0.7430	0.7342	0.7403	0.7382
AUPRC_TRAIN	0.4280	0.4367	0.4445	0.4267	0.4484	0.4387	0.4404	0.4286	0.4328	0.4329
MSE_TEST	0.1130	0.1122	0.1122	0.1134	0.1108	0.1116	0.1119	0.1134	0.1142	0.1126
AUROC_TEST	0.7528	0.7526	0.7513	0.7511	0.7509	0.7523	0.7518	0.7505	0.7462	0.7513
AUPRC_TEST	0.4582	0.4564	0.4561	0.4549	0.4549	0.4542	0.4539	0.4532	0.4523	0.4521
REPLACEOUTLIERS	P025975	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	P025975	P025975
NSAMPLES	253513	253513	253513	253513	253513	253513	253513	253513	253513	253513
SAVEIMAGE	TRUE	TRUE	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE
ELAPSED_MIN	302.54	964.58	318.69	368.98	2119.31	389.56	531.00	398.53	154.68	372.23

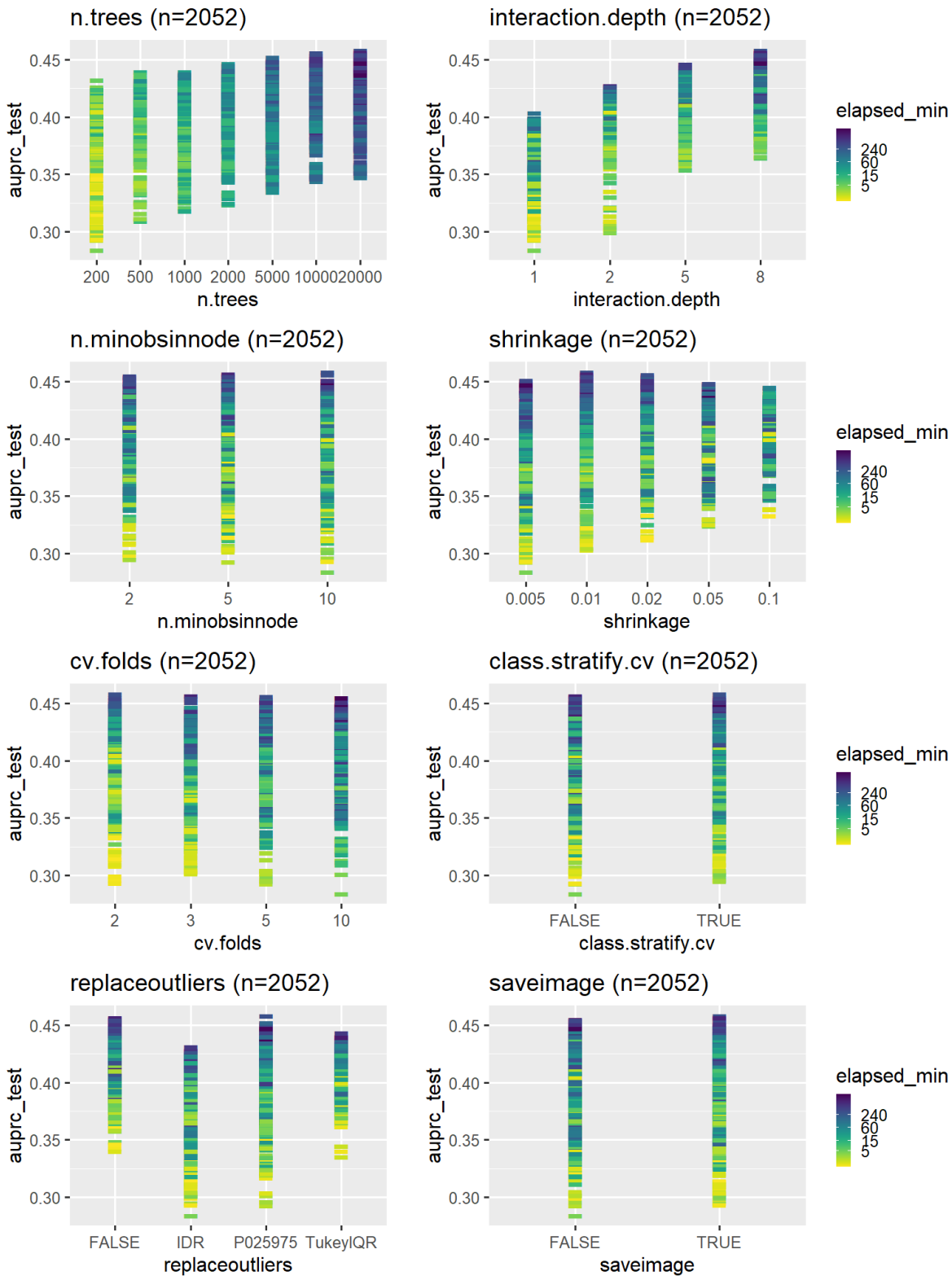

Figure 1. Sensitivity analysis triplots. Cross-validation error and benchmarks for each calibration parameter.

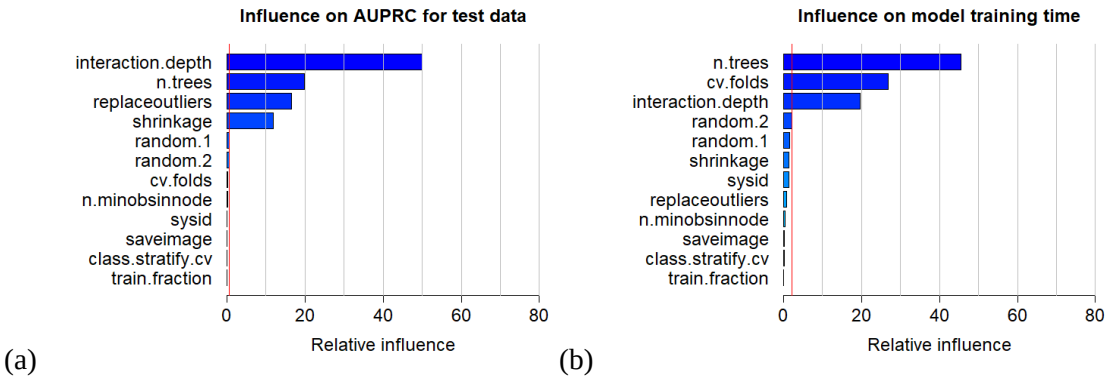

Figure 2. Relative influence ranking of PCL 1.0 GBM parameters on area under precision-recall curve (AUPRC) for test data (a) and model training time (b), determined with separate GBMs (200 trees, full interactions and 10-fold cross-validation) (*n* = 2052 simulations with PCL 1.0 GBM). The noise level is indicated with a red line.

Table 3. PCL 1.0 GBM parameter influences on area under precision-recall curve (AUPRC) for test data and model training time (*n* = 2056 simulations with PCL 1.0 GBM), each determined with a GBM (200 trees, full interactions and 10-fold cross-validation). *Influence below random noise level.

PARAMETER	RELATIVE INFLUENCE ON AUPRC TEST	RELATIVE INFLUENCE ON MODEL TRAINING TIME (BENCHMARK)
N.TREES (GBM)	19.86%	45.42%
INTERACTION.DEPTH (GBM)	50.01%	19.72%
N.MINOBSINNODE (GBM)	0.12%*	0.47%*
SHRINKAGE (GBM)	11.92%	1.42%*
CV.FOLDS (GBM)	0.20%*	26.83%
CLASS.STRATIFY (GBM)	0.03%*	0.14%*
REPLACEOUTLIERS (WRAPPER)	16.56%	0.68%*
SAVEIMAGE (WRAPPER)	0.04%*	0.27%*
SYSID (WRAPPER)	0.05%*	1.27%*
RANDOM.1	0.61%	1.56%
RANDOM.2	0.58%	2.22%
TOTAL	100%	100%

4 RECOMMENDATIONS

4.1 Recommendations for PCL 1.0 GBM parameterization

We now provide some recommendations based on the sensitivity analysis and benchmark test. The optimal values listed in Table 3 apply specifically to the test dataset and PCL 1.0 GBM, and we selected them while prioritizing high AUPRC, and model training time (secondary importance). If a parameter has a GBM relative influence on AUPRC smaller than that of a random variable, we assume its effect on performance is negligible.

The optimal parameter values in Table 3 are to a limited extent applicable to other areas with comparable conditions, but we cannot guarantee success. Values will always vary across regions depending on physical and dynamic characteristics (topography, fuel characteristics, road networks, fire suppression effort, weather, and actual fire perimeter locations) and quality of the data provided to the model.

Table 4. Parameter values for PCL 1.0 GBM optimized for AUPRC and model training speed.

Parameter	Description	Optimal value
n.trees	Total number of trees	Set to 5000. While larger trees (10000 or 20000 trees) can outperform smaller trees in terms of AUPRC in testing mode, there, more improvement can be achieved at a lower computational cost by increasing the interaction depth.
interaction.depth	Levels of interaction	Set equal to the number of predictor variables minus 1 (full interactions). If speed is important, reduce to 5. More interactions improve AUPRC but increase training time.
n.minobsinnode	Minimum number of observations in terminal nodes	Leave at default value (10). This parameter has no appreciable effect on performance or on model training time.
shrinkage	Learning rate or step-size reduction	Leave at default value (0.1); set to 0.02 for n.trees=5000, 10000, 20000
train.fraction	Fraction of observations used to fit the gbm vs out-of-sample estimate	Leave at default value (1); or for train-test split set to 0.7.
cv.fold	Number of cross-validation folds	Set to 2. The purpose of *k*-fold cross-validation is not to improve model accuracy, but to estimate model accuracy at the end of a full run. For smaller sample sizes (n < 500), cross-validation averages out the effect of random sampling on the cross-validation error. But for large sample sizes the effect of random sampling on calculated cross-validation error appears nil, while still increasing the calculation time with the number of folds.
class.stratify.cv	Cross-validation stratify by class	Leave at default value (NULL). This parameter has no bearing on AUPRC (see under cv.fold).

outlierreplace	Outlier processing (wrapper function)	Set to FALSE. None of the methods we programmed in data.table (*viz.* TukeyIQR, IDR and P025975) have a significant effect on AUPRC.
saveimage	Saving a model as an *R* workspace image (wrapper function)	Activate. Saving and compressing a GBM model image as an R workspace with the save.image(compress = "gzip") function has no impact on the benchmark, and should always be done as a matter of reproducibility. Workspaces are highly compressible.

4.2 Recommendations for using the *R* software

R does not handle system resources (CPU, memory etc.) itself, but defers this task to the operating system (*e.g.* Windows, MacOS, Linux). While functions in popular machine learning packages like *gbm* are internally optimized for performance, there are ways to improve how R interacts with the system. In Table 5 we formulate recommendations for faster computing in R that apply to PCL 1.0 GBM and other machine learning tasks.

Table 5. Recommendations for faster computing in *R*.

Priority	Do this in R:	Why this is faster:
1	Use fread() instead of read.csv() and read.table() to import data	fread() in the *data.table* package is the faster option for data reading, and creates an object having both classes data.table and data.frame. The data frame class that R often uses is a list of factors, vectors, and/or matrices. This is an advantage but memory addressing is more complex. Data tables are faster than data frames and dplyr because their structure resembles indexed matrices in relational databases. Because data tables also have the class data.frame, functions like gbm() can accept data tables without the need to modify anything in the function.

2	Use foreach() instead of for() to take advantage of the multicore processing capability	The *doParallel* package and foreach() function (*foreach* package) enable parallel processing of code that can be written as a loop [for() and while() functions]. The following code example uses all cores except one, so that the machine remains available for other tasks:
		`library(doParallel)`
		`library(foreach)`
		`no_cores <- detectCores() - 1`
		`cl <- makeCluster(no_cores)`
		`registerDoParallel(cl)`
		`result <- foreach(i = 1:ncol(mydata)) %dopar% {`
		`  try({`
		`# Code entered here is looped over all columns in mydata`
		`  })`
		`}`
		
		The function try() will always evaluate without error, meaning that foreach() will continue with the next iteration in the loop regardless of whether the code within try() returned an error. Try() will save any error messages to an invisible object of class try-error.

		It is also possible to allow processes running in parallel simultaneous access to the same file by submitting the process ID of the R session when creating a file connection:
		`log_file <- "mylog.csv"`
		`conn <- file(sprintf(log_file, Sys.getpid()), open = "a")`
		`fwrite(x = myresult, conn, append = T)`
		`close(conn)`

3	Use gc() to release unused memory, for example within loops ("for" and "while")	gc() returns unused memory to the operating system. Because the operating system handles the memory for R this reduces memory fragmentation, meaning that objects require fewer memory addresses and can be accessed faster.
4	In RStudio, execute scripts by clicking "Source" instead of "Run" or "Source with Echo"	"Source" executes the entire program at once (fast), while "Source with Echo" and "Run" wait for each message to finish printing before continuing execution (slow).
5	Set "verbose" to FALSE whenever a function has this option	Otherwise, "verbose" waits for each message to finish printing before continuing code execution (slow).

5 ACKNOWLEDGMENT

This work is part of the project "Machine learning for predictive fire planning and fire operations management," funded by USDA Forest Service Rocky Mountain Research Station In-Service Agreement 2216-19-026 with USDA Forest Service Rocky Mountain Research Station USDA Forest Service Southern Research Station. Dennis Hallema designed the sensitivity analysis/benchmark experiment, performed the simulations and wrote the report. Christopher O'Connor designed the PCL gradient boosting model and prepared the data. Dennis Hallema received funding via USDA Forest Service Cost Share Agreement 18-CS-11330110-075 with North Carolina State University. We thank David E. Calkin (USDA Forest Service Rocky Mountain Research Station) and Ge Sun (USDA Forest Service Southern Research Station) for their respective contributions. Any opinions, findings, conclusions or recommendations expressed in this work are those of the authors and do not necessarily reflect the policies and views of the Government of the United States of America. Any use of trade, firm or product names is for descriptive purposes only and does not imply endorsement by the Government of the United States of America.

6 REFERENCES

Elith, J., & Leathwick, J. (2017). Boosted Regression Trees for ecological modeling. R Documentation. Available online: https://cran.r-project.org/web/packages/dismo/vignettes/brt.pdf

Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. Annals of Statistics, 1189-1232.

Friedman, J. H., Hastie, T., and Tibshirani, R. (2000). Additive logistic regression: a statistical view of boosting (with discussion). Annals of Statistics 28, 337-407.

Hallema, D. W., Sun, G., Caldwell, P. V., Norman, S. P., Cohen, E. C., Liu, Y., Bladon, K. D. & McNulty, S. G. (2018). Burned forests impact water supplies. Nature Communications 9: 1307.

Elith, J., Leathwick, J. R., & Hastie, T. (2008). A working guide to boosted regression trees. Journal of Animal Ecology, 77(4), 802-813.

O'Connor, C. D., Calkin, D. E., & Thompson, M. P. (2017). An empirical machine learning method for predicting potential fire control locations for pre-fire planning and operational fire management. International Journal of Wildland Fire, 26(7), 587-597.

R Core Team (2018). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL: https://www.R-project.org/.

Ridgeway, G. (2018). Package 'gbm', Generalized Boosted Regression Models. R Package Version 2.1.4 https://cran.r-project.org/web/packages/gbm/