

Gradient Boosting Regression Trees with Variable Shrinkage

Austin Barket

Department of Computer Science
The Pennsylvania State University at Harrisburg
Middletown, PA 17057
Email: amb6470@psu.edu

Jeremy Blum

Department of Computer Science
The Pennsylvania State University at Harrisburg
Middletown, PA 17057
Email: jjb24@psu.edu

Abstract—In this research we explore the effect of variable shrinkage (learning rate) on gradient boosting machines that utilize regression trees as the base learners. Conventionally, research and implementations of gradient boosting machines have used only constant shrinkage that is applied uniformly to the predictions of all base learners.

I. INTRODUCTION

Boosting is a machine learning technique that combines learning algorithms that barely beat random guessing, known as a weak or base learners, into a single model with significantly improved accuracy or lower error rates over any of its constituent parts [2] [3].

The gradient boosting machine, originally introduced by Friedman in 1999 is a general boosting framework that leverages the steepest descent numerical optimization method to iteratively train base learners to address the errors made by those before them [4].

Production implementations of gradient boosting machines such as the gbm package in R have found remarkable traction among researchers in a wide variety of fields including robotics and ecology [5] [6]. Interestingly these packages generally implement Friedman's Gradient Boost algorithm as it was originally defined [1], leaving some definite opportunity for research into algorithmic tweaks to improve performance.

A particular component of the algorithm that has not been explored to date is the shrinkage, also referred to as shrinkage, which is implemented as a constant parameter to the model. After each iteration the new base learner's predictions are scaled by this parameter as a form of regularization.

This proposal outlines a new way to think about shrinkage for the common case where the base learners are regression trees. We hypothesize that by varying the shrinkage applied to the prediction of the examples in each individual leaf of the regression tree base learners, we will be able to decrease convergence time without sacrificing resilience to overfitting.

II. RELATED WORK

Boosting finds its roots in a question originally posed by Kearns and Valiant in 1988, is weak learnability equivalent to strong learnability [7] [8]? That is if you have a way to learn well enough to beat random guessing, is it inherently true that a strong learner, capable of arbitrarily low error, for that same problem exists? Schapire successfully proved this equivalence in 1990 by proposing and proving the correctness of a polynomial time boosting model he termed *The hypothesis boosting mechanism* [2].

After Schapire's compelling proof that weak and strong learnability are in fact equivalent, researchers began working to improve upon his boosting algorithm. The first practical application of the early boosting algorithms came out of the work of Drucker, Schapire, and Simard at AT&T Bell Labs in 1992. There they applied boosting of neural network base learners to the problem of optical character recognition of handwritten zip codes on USPS letters [9].

In 1995 Freund and Schapire introduced the AdaBoost algorithm which is hailed as solving many of the practical problems suffered by previous boosting algorithms. The unique idea introduced by Adaboost is the notion of applying higher weights at each iteration to the training examples that were misclassified in previous iterations, forcing the new base learners to focus their efforts on these examples. AdaBoost became famous as an excellent out of the bag approach for classification with exceptional resilience to overfitting [10]. However, the details of why exactly AdaBoost worked were unknown until the year 1998 when Friedman, Hastie, and Tibshirani explored the algorithm from an in depth statistical viewpoint. They found that AdaBoost a specialized additive model, and applied knowledge from the long history of statistical additive function approximation to gain a better understanding of AdaBoost and boosting in general [11].

With an increased theoretical statistical understanding of boosting now available, Friedman developed a generalized stagewise additive function approximation boosting model termed the gradient boosting machine in 1999, which he later extended to include a stochastic subsampling approach in 2002 [4] [12]. Gradient Boosting Machines will be explored in great detail in the following sections as extending this model is the focus of the proposed research.

Since their introduction to the machine learning and data mining communities in 1999, gradient boosting machines have found applications in a variety of fields for both classification and regression tasks. Most recently ecology researchers have found great interest in gradient boosting machines, particularly the variant of them that utilizes classification or regression trees as base learners. In 2007, Glenn De'ath extended the R package gbm, creating a new package gbmplus that implements a variant algorithm he terms Aggregated Boosted Trees (ABT). The idea behind ABTs is to perform cross validation to determine an optimal number of iterations for the boosting, then save the models built during cross validation chopping them off at the optimal number of iterations found. To make a prediction, the predictions of all of these boosted trees are computed then averaged. It was found that this approach lead to improved accuracy over gbm alone [13].

Another group of ecological researchers Jane Elith and John Leathwick have also been applying boosted regression trees to their work. One such problem involves predicting whether or not a particular species of eel will be present in unsampled Australian rivers based upon measured environmental factors [6]. Elith and Leathwick implemented their own extensions to the functions in the gbm package in their dismo package in 2015 [14].

III. THE GRADIENT BOOSTING MACHINE

In supervised learning, our goal is to find an approximation \hat{F} of an unknown function $F : \vec{x} \rightarrow y$ that maps data instances \vec{x} to a set of response variables y and best minimizes the expected value of some loss function $\Psi(y, F(\vec{x}))$. Friedman's gradient boosting

machine achieves this by iteratively constructing a strong learner that approximates F from many weak learners. In each iteration a new weak learner, such as a short regression tree, $h(\vec{x})$ is trained to fit the errors made by the function approximation so far. This training is based upon an extremely common numerical minimization method known as steepest gradient descent [1] [4]. However, unlike most applications of steepest descent, Friedman's Gradient Boost algorithm computes the negative gradient \vec{g} in the space of the estimated function itself, not in the space of a finite set of parameters that define the function. By framing the problem in this way, the function \hat{F} is not limited to a set of functions definable by a finite set of parameters, but rather is defined by a potentially infinite set of parameters, one for each possible value \vec{x} . Obviously, it is impossible to actually compute the gradient and apply steepest-descent in this potentially infinite dimensional function space, but it is possible to perform steepest-descent with respect to the finite space of training examples D [1] [4].

The negative gradient in this restricted subset of function space defines the direction of steepest descent in the loss function for the training examples. Thus by updating the function \hat{F} directly by this negative gradient, we would move closer to the minimum values of the loss function Ψ for the examples in the training dataset. Of course this is not quite the goal, instead we would like to be able to generalize to all possible data. To accomplish this we instead train a regression model to predict the negative gradient of the loss function at each step, then update our function with this model's prediction. Friedman's general Gradient Boosting Machine, extended to include his later ideas of subsampling the training data and applying a constant shrinkage to improve generalization is provided in Algorithm 1 [1] [4] [12].

IV. SIMPLIFYING ASSUMPTIONS

For the purposes of the current research, we will consider only the case where Ψ is the squared error function and the goal is to predict a real valued response variable. Without loss of generality we will add a coefficient of $\frac{1}{2}$ to Ψ so that the negative of the partial derivative of Ψ with respect to the predicted value of an instance \vec{x}_i is simply the residual of that prediction. In this case the component wise calculation of the negative gradient (Equation 3) becomes Equation 7.

$$g_{m,i} = -\frac{\partial}{\partial \hat{F}_{m-1}(\vec{x}_i)} \frac{(y_i - \hat{F}_{m-1}(\vec{x}_i))^2}{2} = (y_i - \hat{F}_{m-1}(\vec{x}_i)) \quad (7)$$

As alluded to in the introduction, the proposed variable shrinkage scheme is specially designed for the common case where the base learners are regression trees, we define the notation for regression trees in Figure 1 below.

When building the trees, we define the next best split as the split that leads to the largest reduction in squared error on the training examples. Thus the least squares coefficient β in Equation 4 will always be 1, and since we're using the squared error loss function so will the optimal coefficient ρ in Equations 5 and 6.

Thus, for the case where the base learners are regression trees and the loss function is the squared error, the update step in Equation 6 can be replaced by Equation 8

$$\hat{F}_m(\vec{x}) = \hat{F}_{m-1}(\vec{x}) + v \cdot \sum_{j=1}^J b_{m,j} I(\vec{x} \in R_{m,j}) \quad (8)$$

The choice of regression tree base learners introduces two additional parameters to the gradient boosting algorithm, namely the

input : Training Dataset: $D = (x_i, y_i), i = 1 \dots N$
 Bag Fraction: $bf \in [0, 1]$
 Shrinkage: $v \in [0, 1]$
 Number of Base Learners: M
 Loss Function: Ψ e.g. RMSE
 Choice of Base Learner: $h(\vec{x})$ e.g. regression trees

output: A function $\hat{F}(\vec{x})$ that minimizes the expected value of $\Psi(y, F(\vec{x}))$

Initialize the approximation of \hat{F}

$$\hat{F}_0(\vec{x}) = \underset{\rho}{\operatorname{argmin}} \sum_{i=1}^N \Psi(y_i, \rho) \quad (1)$$

for $m \leftarrow 1$ **to** M **do**

Select a random subsample, S_m , of size \tilde{N} from the training data without replacement.

$$S_m \subset D, \quad |S_m| = \tilde{N} = bf \cdot N \quad (2)$$

Approximate the negative gradient \vec{g}_m of $\Psi(y_i, \hat{F}_{m-1}(\vec{x}))$ with respect to $\hat{F}_{m-1}(\vec{x})$.

$$g_{m,i} = -\frac{\partial}{\partial \hat{F}_{m-1}(\vec{x}_i)} \Psi(y_i, \hat{F}_{m-1}(\vec{x}_i)), \quad \vec{x}_i \in S_m \quad (3)$$

Train a new base learner $h_m(\vec{x})$ to predict \vec{g}_m and fit the least squares.

$$\beta_m, h_m(\vec{x}) = \underset{\beta, h(\vec{x})}{\operatorname{argmin}} \sum_{\vec{x}_i \in S_m} [g_{m,i} - \beta h(\vec{x}_i)]^2 \quad (4)$$

Solve for the optimal coefficient ρ that minimizes Ψ .

$$p_m = \underset{\rho}{\operatorname{argmin}} \sum_{\vec{x}_i \in S_m} \Psi(y_i, \hat{F}_{m-1}(\vec{x}_i) + \rho h(\vec{x}_i)) \quad (5)$$

Update your approximation of \hat{F} , scaled by the shrinkage v

$$\hat{F}_m(\vec{x}) = \hat{F}_{m-1}(\vec{x}) + v \cdot \rho_m h_m(\vec{x}) \quad (6)$$

end

Algorithm 1: Friedman's Gradient Boost Algorithm [4] [5] [12] [13]

maximum number of splits in each tree (aka interaction depth) and the minimum number of observations in each leaf node. These parameters work together to regularize the complexity of the regression trees, and in general these parameters are chosen to ensure short trees and stout leaves to weaken the predictive power of each individual base learner which ultimately strengthens the boosted model's ability to generalize [6].

Note that although we restrict the current study to regression tasks using the squared error loss function, the variable shrinkage scheme defined and tested in the following sections could easily be applied to any of the loss functions originally defined by Friedman in [4] [12], and most famously implemented in the gbm R package originally written by Ridgeway [1]. Please see these references for information on the many specialized derivations of Algorithm 1 for various learning tasks and to better understand the reasoning behind the equations defined in this section.

$$h_m(\vec{x}) = \sum_{j=1}^J b_{m,j} I(\vec{x} \in R_{m,j}) \quad (9)$$

Where

J = the number of terminal nodes (leaves) in the tree

$b_{m,j}$ = Prediction made for all instances in $R_{m,j}$.

For squared error, $b_{m,j} = \text{avg}_{\vec{x} \in R_{m,j}} (g_{m,i})$

$R_{m,j}$ = The subset of instances $\vec{x} \in S_m$ that are predicted by the j^{th} terminal node.

$$I(\alpha) = \begin{cases} 1 & \alpha \text{ is true} \\ 0 & \alpha \text{ is false} \end{cases}$$

Fig. 1: Notation for Regression Tree Base Learners

V. VARIABLE SHRINKAGE FOR REGRESSION TREE BASE LEARNERS

The use of regression trees presents an interesting possibility of a simple, yet elegant adaptation method. Since the trees themselves can be seen as a summation of individual prediction terms, one for each leaf, a natural adaptation scheme is to compute a different shrinkage for each leaf node in each base learner. Specifically a simple linear mapping can be used to ensure that the lower the number of examples in a given leaf node, the lower the computed shrinkage. We hypothesize that this will encourage the training to learn rapidly from the most typical training examples, which will fall into large leaf nodes along with their similar peers, while diminishing the impact of outlying and noisy examples, which will be isolated by the regression tree's splitting algorithm. Overall we expect this scheme to lead to faster convergence time while strongly discouraging overfitting.

To this end we will alter Algorithm 1 to take as input both a minimum and maximum shrinkage v_{\min} and v_{\max} , instead of the constant shrinkage v . The following equation will then be used to compute the shrinkage for each of the J leaves in Equation 8.

$$v_{m,j} = \frac{|R_{m,j}|}{|S_m|} (v_{\max} - v_{\min}) + v_{\min} \quad (10)$$

This equation maps the range of possible leaf node sizes $[0, \tilde{N}]$ to the range of possible shrinkage values $[v_{\min}, v_{\max}]$. Note that even when the regression trees are built with a nonzero minimum number of observations in each leaf node, the missing value branches of the trees can still have any number of examples in them, including zero if no missing values existed in the training data. Thus zero is still used as the minimum leaf size for the purposes of shrinkage calculation regardless of the minimum number of observations parameter.

Using equation 10, the update step (Equation 8) becomes...

$$\hat{F}_m(\vec{x}) = \hat{F}_{m-1}(\vec{x}) + \rho_m \sum_{j=1}^J v_{m,j} \cdot b_{m,j} I(\vec{x} \in R_{m,j}) \quad (11)$$

VI. IMPLEMENTATION

We implemented the Friedman's gradient boosting machine with the option to utilize either the standard constant shrinkage or our proposed variable shrinkage scheme as defined in Sections III, IV, and V. The core algorithm implementation is an extension of the very minimal JSGBM project found here [15]. The additions to

the original project include support for missing values and splits on categorical variables, optimization of the splitting algorithm, calculation of predictor relative influence, a cross validation based method for selecting the optimal number of trees, and support for De'ath's aggregated boosted trees as discussed in Section II. All code implemented for this paper can be found at [16].

The algorithm for selecting the optimal number of trees is based on the `gbm.step` function of the `dismo` R package which is described in [6]. The pseudocode for our implementation is provided in Algorithm 2.

input : Number of Folds: $K = 4$
Step Size: $S_{\text{size}} = 5,000$;
Minimum Error Delta: $\Delta_{\min} = 10^{-6} \cdot \text{meanResponse}$
Steps Past Minimum: $S_{\text{pm}} = 3$
Max Number of Trees: $M = 150,000$

output: Estimated number of trees at which generalization error is minimized

Create K training/validation set pairs as in normal K-fold cross validation. For each pair, create a new empty GBM model.

Initialize number of steps remaining

$$S_{\text{remaining}} \leftarrow S_{\text{pm}}$$

while NumberOfTrees $< M$ and $S_{\text{remaining}} > 0$ **do**
Add S_{size} trees to each of the K GBM models.

Evaluate the new validation RMSE of each model, and average them.

$$\Delta_{\text{error}} \leftarrow \text{oldAvgRMSE} - \text{newAvgRMSE}$$

if $\Delta_{\text{error}} > \Delta_{\min}$ **then**

$$S_{\text{remaining}} \leftarrow S_{\text{pm}}$$

else

$$S_{\text{remaining}} \leftarrow S_{\text{remaining}} - 1$$

end

Algorithm 2: Find Optimal Number of Trees using Cross Validation

VII. DATASETS

Four real world datasets with natural regression tasks were selected for use in this study. Each of these can be found in the UC Irvine Machine Learning Repository [17]. This section introduces each of these in turn.

A. Combined Cycle Power Plant (powerPlant)

This dataset contains 9568 examples collected over 6 years from a power plant set work at full load. Each instance consists of 4 real valued predictors; namely the Temperature (T), Ambient Pressure (AP), Relative Humidity (RH) and Exhaust Vacuum (V). The target response variable is the net hourly electrical energy output (EP) of the plant [18].

B. Airfoil Self-Noise (*nasa*)

This data originated from a 1989 NASA experiment in which different size air foils were placed in a wind tunnels and subjected to various free-stream velocities and angles of attack. The regression goal is to predict the scaled sound pressure level in decibels of an air foil given the frequency (Hertz), angle of attack (degrees), chord length (meters), free-stream velocity (meters/second), and suction side displacement thickness (meters) [19].

C. Bike Sharing (*bikeSharing*)

Hadi Fanaee from the Laboratory of Artificial Intelligence and Decision Support (LIAAD), University of Porto pulled together weather information and holiday information and integrated it with publically available trip data provided by Capital Bikeshare to generate this intriguing dataset consisting of almost two years of bike rental data aggregated both by day and by hour. Each instance contains attributes providing information about the day [and hour] that the rentals occurred. This includes the year, season, month, day, hour, whether it was a holiday, a weekday, or a working day. In addition the weather conditions for each instant in time are provided including the precipitation type, temperature, humidity, and wind speed. Finally the count of bikes rented during each time period is provided, which will be used as the target variable in our experiment [20] [21].

D. Communities and Crime (*crimeCommunities*)

Lastly we will look at the Communities and Crime dataset which "combines socio-economic data from the 1990 US Census, law enforcement data from the 1990 US LEMAS survey, and crime data from the 1995 FBI UCR." [?]

VIII. PARAMETER TUNING EXPERIMENT

In order to compare model performance of the original algorithm with our proposed shrinkage adaption scheme, we developed an experiment in which 2048 total sets of parameters were tested on the four datasets described in Section VII.

All parameters that affect the execution of Algorithm 2, which is the core training procedure used in all of our tests, are shown in Table I. Two additional parameters are listed that were not introduced with algorithm itself but are important to node. These indicate that the running time and memory usage for during training was limited to 1.5 hours and 20 GB respectively. These values were exceeded only a handful of times when using the lowest constant learning rate and the highest number of splits. In the rare case that a test exceeded these limits the training stopped and the application continued with collecting and saving metadata about the constructed models, just as though the training stopped due to the normal stopping condition.

Gradient boosting machines with regression tree base learners are traditionally defined by the number of trees to build and 4 additional parameters; the bag fraction, shrinkage, minimum number of examples in each leaf, and the maximum number of splits in each tree. To support our variable shrinkage scheme we must add one additional parameter since we require both a minimum and a maximum learning rate. All values that were considered for each of these parameters are provided in Table II. Taking all possible combinations of these parameters leads to 512 parameter sets with constant shrinkage, and 1536 possibilities with variable shrinkage for a total of 2048 individual tests. For each combination of parameters on each dataset five independent runs were performed and all data collected from them was averaged. All results presented and discussed in the following sections are derived from these average values. Each

Maximum Running Time	1.5 hours
Maximum Memory Usage	20 GB
Maximum Number of Trees	150,000
Number of CV Folds	4
CV Step Size	5000
Number of CV Steps after Falling Below Min Error Delta	3
Minimum Error Delta	0.000001 · mean training response

TABLE I: Execution limits and Parameters for Finding Optimal Number of Trees

Bag Fraction	0.25, 0.50, 0.75, 1
Min Examples In Node	1, 50, 100, 250
Max Number Of Splits	1, 2, 4, 8, 16, 32, 64, 128
Constant: Shrinkage	0.1, 0.01, 0.001, 0.0001
Variable: Minimum Shrinkage	0.01, 0.001, 0.0001
Variable: Maximum Shrinkage	0.1, 0.4, 0.7, 1

TABLE II: Execution limits and Parameters for Finding Optimal Number of Trees

run of a particular parameter set on a dataset consists of the following procedure.

- 1) Select randomly without replacement 20% of the dataset and set aside as a test set.
- 2) Use Algorithm 2 to perform 4-fold cross validation and find the optimal number of trees. Simultaneously train a GBM using the entire training set.
- 3) Build an Aggregated Boosted Tree from the 4 GBMs built for cross validation.
- 4) For each number of trees from 1 to total number of trees built, collect the following information to generate error curves and other interesting graphs. (This is done as steps 2 and 3 execute)
 - a) Average training, validation, and test RMSE of the 4 cross validation GBMs.
 - b) Training and Test RMSE of the GBM trained using all training data.
 - c) Training and Test RMSE of the Aggregated Boosted Tree.
 - d) Mean and Standard Deviation of the actual number of examples across the leaf nodes in the last tree that was added to the all training data GBM.
 - e) Mean and Standard Deviation of the actual shrinkage values across the leaf nodes in the last tree that was added to the all training data GBM.
 - f) The actual number of splits in the last tree that was added to the all training data GBM.
- 5) For each example in the dataset (both training and test) collect the following information.
 - a) Prediction at the optimal number of trees using the all training data GBM.
 - b) The residual of that prediction
 - c) The avg learning rate applied to that example across all trees using the all training data GBM.
- 6) Compute the relative influences of the predictors for both the all training data GBM and the aggregated boosted tree.
- 7) Compute the average and standard deviation of the actual number of splits across all trees built. (Average and Std. Dev. of 4.f)
- 8) Compute the average and standard deviation of the average learning rates applied to each example. (Average and Std. Dev. of 5.c)

write
some
thing

- 9) Record the total amount of time taken to perform all of these steps.

IX. RESULTS

Results and Stuff

X. ANALYSIS

Analysis and stuff

XI. CONCLUSION

The conclusion goes here.

ACKNOWLEDGMENT

Yo chang's the man

REFERENCES

- [1] G. Ridgeway, "Generalized boosted models: A guide to the gbm package," *Update*, vol. 1, no. 1, p. 14, 2012.
 - [2] R. E. Schapire, "The strength of weak learnability," *Mach. Learn.*, vol. 5, no. 2, pp. 197–227, Jul. 1990. [Online]. Available: <http://dx.doi.org/10.1023/A:1022648800760>
 - [3] —, "The boosting approach to machine learning: An overview," in *Nonlinear Estimation and Classification*, ser. Lecture Notes in Statistics, D. Denison, M. Hansen, C. Holmes, B. Mallick, and B. Yu, Eds. Springer New York, 2003, vol. 171, pp. 149–171.
 - [4] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *The Annals of Statistics*, vol. 29, no. 5, pp. pp. 1189–1232, 2001. [Online]. Available: <http://www.jstor.org/stable/2699986>
 - [5] A. Natekin and A. Knoll, "Gradient boosting machines, a tutorial," *Frontiers in Neurorobotics*, vol. 7, p. 21, 2013. [Online]. Available: <http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3885826/>
 - [6] J. Elith, J. R. Leathwick, and T. Hastie, "A working guide to boosted regression trees," *Journal of Animal Ecology*, vol. 77, no. 4, pp. 802–813, 2008. [Online]. Available: <http://dx.doi.org/10.1111/j.1365-2656.2008.01390.x>
 - [7] M. Kearns, "Thoughts on hypothesis boosting," *Unpublished manuscript (Machine Learning class project, December 1988)*, 1988.
 - [8] M. Kearns and L. G. Valiant, "Cryptographic limitations on learning boolean formulae and finite automata," in *Proceedings of the Twenty-first Annual ACM Symposium on Theory of Computing*, ser. STOC '89. New York, NY, USA: ACM, 1989, pp. 433–444. [Online]. Available: <http://doi.acm.org/10.1145/73007.73049>
 - [9] H. Drucker, R. E. Schapire, and P. Simard, "Improving performance in neural networks using a boosting algorithm," in *Advances in Neural Information Processing Systems 5, [NIPS Conference]*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993, pp. 42–49. [Online]. Available: <http://dl.acm.org/citation.cfm?id=645753.668055>
 - [10] Y. Freund and R. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," in *Computational Learning Theory*, ser. Lecture Notes in Computer Science, P. Vitnyi, Ed. Springer Berlin Heidelberg, 1995, vol. 904, pp. 23–37. [Online]. Available: <http://dx.doi.org/10.1007/3\540\59119-2\166>
 - [11] J. Friedman, T. Hastie, and R. Tibshirani, "Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors)," *Ann. Statist.*, vol. 28, no. 2, pp. 337–407, 04 2000. [Online]. Available: <http://dx.doi.org/10.1214/aos/1016218223>
 - [12] J. Friedman, "Stochastic gradient boosting," *Computational Statistics And Data Analysis*, vol. 38, no. 4, pp. pp 367–378, 2002. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0167947301000652>
 - [13] G. De'Ath, "Boosted trees for ecological modeling and prediction," *Ecology*, vol. 88, no. 1, pp. 243–251, 2007.
 - [14] J. Elith and J. Leathwick, "Boosted regression trees for ecological modeling," 2015.
 - [15] "Jsgbm," [Online]. Available: <https://code.google.com/p/jsgbm/>
 - [16] "Project implementation," [Online]. Available: <https://github.com/ambarket/GBMWithVariableShrinkage>
 - [17] M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: <http://archive.ics.uci.edu/ml>
 - [18] "Combined cycle power plant data set," [Online]. Available: <http://archive.ics.uci.edu/ml/datasets/Combined+Cycle+Power+Plant>
 - [19] "Airfoil self-noise data set," [Online]. Available: <http://archive.ics.uci.edu/ml/datasets/Airfoil+Self-Noise>
 - [20] "Bike sharing dataset data set," [Online]. Available: <http://archive.ics.uci.edu/ml/datasets/Bike+Sharing+Dataset>
 - [21] H. Fanaee-T and J. Gama, "Event labeling combining ensemble detectors and background knowledge," *Progress in Artificial Intelligence*, pp. 1–15, 2013. [Online]. Available: [WebLink]
- U. S. Department of Commerce, Bureau of the Census, Census Of Population And Housing 1990 United States: Summary Tape File 1a & 3a (Computer Files),
- U.S. Department Of Commerce, Bureau Of The Census Producer, Washington, DC and Inter-university Consortium for Political and Social Research Ann Arbor, Michigan. (1992)
- U.S. Department of Justice, Bureau of Justice Statistics, Law Enforcement Management And Administrative Statistics (Computer File) U.S. Department Of Commerce, Bureau Of The Census Producer, Washington, DC and Inter-university Consortium for Political and Social Research Ann Arbor, Michigan. (1992)
- U.S. Department of Justice, Federal Bureau of Investigation, Crime in the United States (Computer File) (1995)
- Redmond, M. A. and A. Baveja: A Data-Driven Software Tool for Enabling Cooperative Information Sharing Among Police Departments. *European Journal of Operational Research* 141 (2002) 660-678.