PENN STATE HARRISBURG
THE CAPITAL COLLEGE HONORS PROGRAM

SCHOOL OF SCIENCE, ENGINEERING, AND TECHNOLOGY

ANALYZING THE EFFECT OF NOISE INJECTION ON GRADIENT BOOSTING MACHINE REGRESSION MODELS

HAMZA BACHNAK
Fall 2023

A thesis
submitted in partial fulfillment
of the requirements
for baccalaureate degree
in Computer Science
with honors in Computer Science

Reviewed and approved* by the following:

Jeremy Joseph Blum
Associate Professor of Computer Science, School of Science, Engineering, and Technology
Program Chair, Computer Science & Mathematics
Thesis Supervisor

Md Faisal Kabir
Assistant Professor of Computer Science, School of Science, Engineering, and Technology
Faculty Reader

Thang N. Bui
Associate Director, School of Science, Engineering, and Technology
Associate Professor of Computer Science, School of Science, Engineering, and Technology
Honors Adviser

* Signatures are on file in the Capital College Honors Program

Abstract

Decision trees are a widely used approach for regression tasks. Much like its name suggests, a decision tree resembles a tree, beginning from a single root node, splitting off into different branch nodes, and finally ending on leaf nodes. Typically, for a regression task, a single value in the leaf node is used for a prediction. The discrete nature of trees results in outputs that are discontinuous. In gradient boosting machines (GBMs), multiple decision trees are then bundled together to produce output. We hypothesize that one reason for the large number of trees in GBM models is the discontinuous nature of tree-based predictions, and therefore, if other means were used to smooth out GBM predictions, smaller models could produce similar results. This research investigates the ability of noise injected into input vectors during the prediction to allow for smaller GBM model, potentially reducing runtime and memory usage, without sacrificing model accuracy. The test using a smaller GBM model with injected randomness did not exhibit an increase in accuracy. However, a parallel exploration with a similar-sized model incorporating randomness yielded promising results, demonstrating improved accuracy under certain conditions. This highlights the intricate relationship between model size, randomness, and predictive performance in GBMs. It is crucial to acknowledge some limitations encountered during our research. The handling of categorical data posed challenges, as GBMs may not handle such data well in the presence of noise. Additionally, compatibility issues within the GBM framework and complexities associated with constraints presented obstacles that need to be addressed for a more comprehensive understanding. Despite these limitations, our study provides insights into the benefits and challenges associated with introducing randomness into GBM models. Further refinements of the proposed methodology are needed to address the identified limitations and optimize the use of noise injection for enhanced efficiency and accuracy.

Table of Contents

List of Figures

List of Tables

Acknowledgements

I would like to extend my heartfelt gratitude to my thesis advisor, Dr. Jeremy Blum, for his unwavering support, guidance, and invaluable expertise throughout the research and writing process of this thesis. His mentorship and dedication have been instrumental in shaping my ideas and helping me navigate the challenges that came my way. His insightful feedback, constructive criticism, and thoughtful suggestions have contributed significantly to the refinement of this work.

I would also like to express my appreciation to Dr. Faisal Kabir, who served as the reader for this thesis. Their willingness to invest time in reviewing this research is deeply appreciated.

Furthermore, I would like to thank Dr. Shirley Clark, Dr. Gina Brelsford, and Ms. Stephanie Ponnett for their guidance throughout the thesis shaping and writing processes.

Last but certainly not least, I want to thank my family for their unwavering encouragement, understanding, and love. Their constant support and belief in my abilities have been the cornerstone of my academic journey. I am profoundly grateful for their patience during long hours of research and writing and for being a source of motivation and strength.

To all of you, I owe the successful completion of this thesis. Your support, wisdom, and encouragement have been indispensable, and I am truly thankful for the roles you've played in this academic endeavor.

Chapter 1: Introduction

One of the more heavily utilized tools in machine learning is the decision tree algorithm. In the decision tree shown in Figure 1, we note the new cases of larynx cancer within different age groups of different sexes using data provided by Center of Disease Control [1]. The decision tree indicates the factor of age to display various odds of contracting larynx cancer.

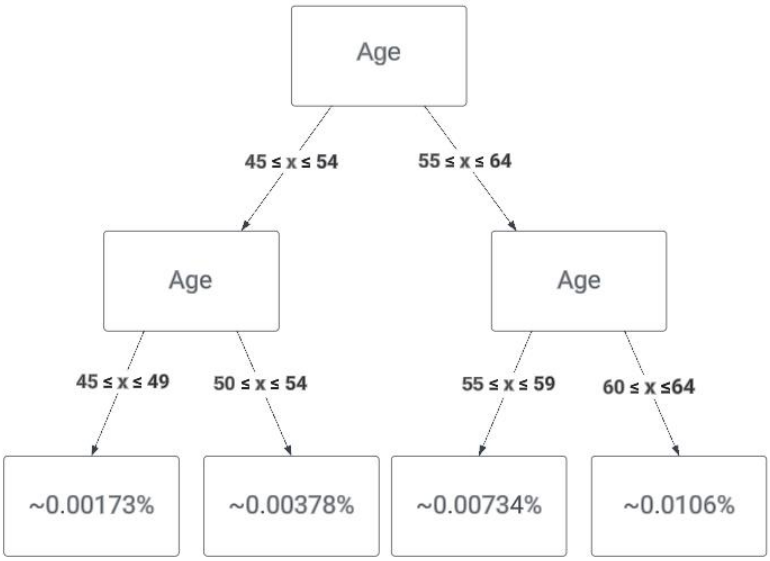

Figure 1: Rates of new larynx cancers by age reported by CDC **[1]**

To use the decision tree, shown above, we would look at the age group we are currently targeting. If the age of the individual is within the range of age group 1 (45 <= x <= 54), then we would proceed down the left branch. Otherwise, we would see if the age of the individual satisfies age group 2's age range and proceed down the right branch, if so. This process is repeating till we reach a leaf node containing a probability value.

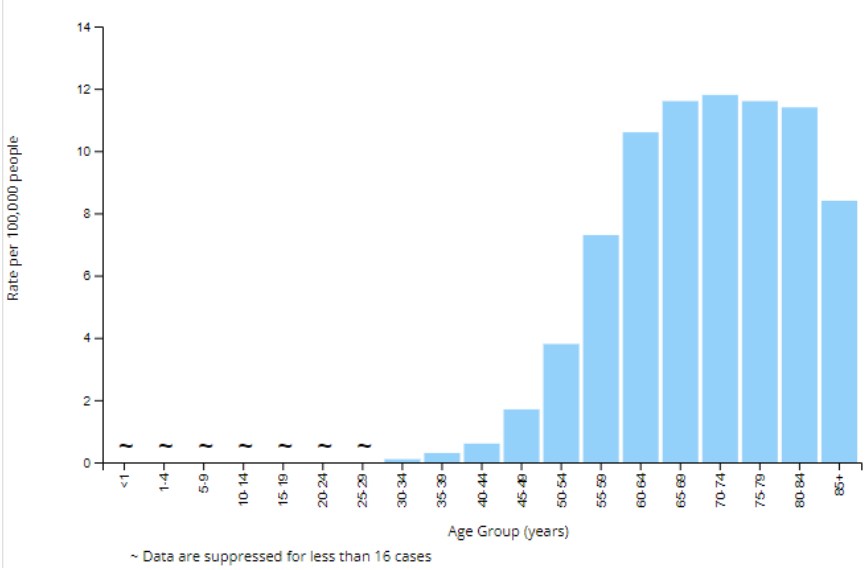

Figure 2: Rates of new larynx cancers by age reported by CDC represented in bar graph **[1]**

Looking closely at the graph shown in Figure 2, we can observe a large discrepancy of being diagnosed with larynx cancer between ages 54 and 55 with a drastic increase of nearly double. This drastic increase in probability leads to discontinuities when interpreting such data. These discontinuities within data can lead to more extreme gaps between variables with close values when producing prediction models and thus a less accurate model.

To help combat such gaps within data, different types of decision tree-base models have been utilized within machine learning. Techniques such as Gradient Boosting Machines (GBM's) and Random Forest both create vast forests of trees. Other techniques such as Linear Model apply linear regression models in the leaves of produced linear trees. Each of these approaches sacrifice computational runtime, the size and scale of the produced models, or both.

We hypothesize that one reason for the large number of trees in GBM models is the discontinuous nature of tree-based predictions, and therefore, if other means were used to

smooth out GBM predictions, smaller models could produce similar results. This hypothesis is tested by training a baseline GBM model using lightGBM followed by using the same model with Gaussian noise added to the input vectors with a goal of smoothing out the predictions. The accuracy of the smoothed predictions is then compared to the baseline GBM. For this research, we will be analyzing datasets of used US cars [2].

We had found a decrease in root mean squared error (RSME), when we applied noise vectors to a model on a synthetic dataset, but an increase in RSME when applied to our real-life dataset of used cars. This result suggests a need for further research within this topic, showing the merit of noise injection to potentially improve accuracy, runtime, or memory of GBM models.

Chapter 2: Literature Review

2.1 Introduction to Machine Learning

Efficiency and speed are two of the most prominent factors when it comes to performance in this age of technology. To facilitate tasks to reach peak performance, people create sets of instructions, formulas, or algorithms. The dictionary of Britannica defines an algorithm as "a specific procedure for solving a well-defined computational problem [3]." Within the field of computer science, algorithms are necessary to help create working formulas and functions to conduct tasks, from artificial intelligence to databases, security programs, and so forth. They also help us analyze why certain methods are able to outperform others.

As technology has advanced in recent years, algorithms and complex systems have taken advantage of available resources. One field that has spawned by such technology and has grown exponentially is Machine Learning, as many different types of algorithms have been developed in this field. Over the years, different attempts have been made to develop more efficient algorithms to help produce models, to be applied to systems.

2.2 Regression Versus Classification

Before delving into specific types of Machine Learning algorithms, an important detail to note is the type of tasks the algorithm is supposed to accomplish. Such tasks may be split into two groups: Regression and Classification. Classification algorithms are designed to map values to discrete categories, assigning them to a predefined label, thus categorizing them. An example of classification may be that of labeling produce as fruit or vegetable, based on variables such as seeds, roots, leaves, etc. Regression, on the other hand, is the process of mapping data to find a

relationship between the variables to produce a model that may predict continuous outcomes. An example of regression may be that of predicting the average amount of rainfall to occur in a week based on prior data and current variables such as humidity and temperature. The major distinction between both classification and regression is that classification algorithms provide a discrete label, while regression algorithms provide an output of continuous values [4-7] [4] [5] [6] [7].

2.3 Decision Trees

Decision trees are one of the more utilized methods used in machine learning to help train new models using provided data sets. Much like its name infers, a decision tree resembles a tree, beginning from a single root node, splitting off into different branch nodes, and finally ending on leaf nodes. A simple example of a decision tree would be a question of: "Is it currently raining?" If the answer in the example is a yes, the decision tree would proceed to a leaf node that may provide a response of "bring an umbrella," otherwise, the tree will proceed to another leaf node with a conclusion of "no umbrella needed". A decision tree does not have to be binary but such trees tend to be the most common in machine learning [8].

2.4 Gradient Boosting Machines

Ensemble learning is the combination of different models to produce a better performing model. The Gradient Boosting Model (GBM) is an approach that makes use of ensemble learning by training several models sequentially. GBM then predicts errors or outstanding values of initial models and sums up to find a starting model. Afterwards, one weak learner is added at a time to the model and previous learners are kept unchanged. GBM soon analyzes and weighs the patterns displayed and strengthens the model accordingly, based on the weak

predictions. The modeling is stopped once the GBM cannot locate any pattern to the model [9-11] [9] [10] [11].

2.5 Bagging, Random Forest, and Boosting Algorithms

Bagging is a type of ensemble learning method that combines results by taking the mean output to produce a prediction. The algorithm can be split into three key steps:

1) Randomly sampled datasets of original training data (bootstrapping).
2) Build and fit several classifiers to each of the diverse copies.
3) Take the average of all predictions to produce result.

Random forest is another ensemble learning method in which multiple random decision trees from a set combine to produce a resulting, more generalized model. Finally, and the most relevant ensemble learning method to this project, is the boosting method. The basic concept behind boosting is correcting errors of previous models and using them to train newer models based off the more improved pattern [9], [11].

2.6 Data Gaps and Outliers

Within datasets, two common issues that require careful consideration are data gaps and outliers. These issues, if not addressed appropriately, can significantly impact the reliability and accuracy of machine learning models.

2.6.1 Data Gaps

Data gaps, sometimes referred to as missing or incomplete data, are instances in which certain data points or attributes within a dataset are unavailable or incomplete. The presence of

data gaps can be attributed to a variety of factors, including technical limitations, human error, or irregular data collection frequencies. The consequences of data gaps in machine learning encompass the following:

- Bias Introduction: Data gaps may lead to bias in the model as it leans towards the information that is present. This can result in skewed predictions and diminished model accuracy.
- Reduced Model Performance: Incomplete datasets can impede the model's ability to recognize patterns, thereby compromising its overall performance in terms of predictive accuracy.
- Inaccurate Insights: The absence of essential data can lead to faulty insights, which could potentially influence decision-making processes in fields such as healthcare, finance, and manufacturing.

2.6.2 Outliers

Outliers are deviations within data that vary significantly from the other data. These deviations can arise from errors, anomalies, or genuine but rare occurrences. The presence of outliers can have a large influence on machine learning models:

- Model Sensitivity: Outliers can disproportionately impact the model's parameters, causing it to be less robust and less capable of generalizing well to new data.
- Inaccurate Predictions: If outliers are not appropriately addressed, they can result in inaccuracies, particularly in regression tasks where the model's predictions may be swayed by these extreme values.

- Overfitting Risk: Machine learning models may tend to overfit when attempting to accommodate outliers, leading to a decline in performance on unseen data.
- Outlier Detection: Methods for identifying outliers include statistical approaches (e.g., z-scores), data visualization tools (e.g., box plots), and machine learning-based algorithms (e.g., isolation forests or one-class SVM).
- Outlier Handling: Once identified, outliers can be managed through strategies such as removal, transformation, or the use of robust modeling techniques that are less sensitive to extreme values.

The approach to dealing with data gaps and outliers should be chosen with careful consideration of the specific characteristics of the dataset, the project's objectives, and the domain of application.

All of these reduce legibility and effectiveness of the models produced [9], [12], [13].

2.7 Prior Attempts

Prior attempts to smoothen a curve, without overfitting data, using both greedy and non-greedy methods have been used [14]. Most of these methods tend to focus on the processing phase of data while using known and working pre-processing techniques to help produce both a training set of data as well as test set of data [8], [12], [13], [15].

2.7.1 Greedy Methods

Greedy methods in machine learning often involve making locally optimal choices at each step with the hope of reaching a globally optimal solution. In the context of smoothening a curve without overfitting, greedy methods might involve iteratively adjusting model parameters to minimize errors. However, the challenge lies in ensuring that the optimization process doesn't

lead to overfitting, where the model becomes too tailored to the training data, losing its ability to generalize to new data [5], [9], [16].

Common techniques within the greedy approach may include:

- Stepwise Regression: A method that sequentially adds or removes predictors to find the subset of variables that optimally fits the model.
- Feature Selection Algorithms: Algorithms that iteratively select or eliminate features based on their impact on model performance.

2.7.2 Non-Greedy Methods

Non-greedy methods, on the other hand, consider the broader context improve system performance. These methods often involve a holistic approach to modeling, considering the overall structure of the data. The challenge is to achieve the desired smoothness without sacrificing the model's ability to capture essential patterns [9], [12].

Notable non-greedy techniques may include:

- Regularization Techniques: Methods like L1 and L2 regularization introduce penalty terms to the model's optimization process, discouraging overly complex models and promoting smoother solutions.
- Kernel Smoothing Techniques: These techniques, such as the Nadaraya-Watson kernel regression, aim to estimate a smooth curve by giving more weight to nearby data points, thus reducing the impact of outliers.
- Cross-Validation Strategies: Employing techniques like k-fold cross-validation helps assess model performance on multiple subsets of the data, providing a more robust evaluation and preventing overfitting to specific data configurations.

2.7.3 Summary

In summary, previous attempts to balance curve smoothness and prevent overfitting have explored a spectrum of approaches, from the locally optimized, greedy methods to the globally informed, non-greedy methods. The choice between these methods often depends on the specific characteristics of the data and the desired trade-off between complexity and generalization in the model.

Chapter 3: Methods

3.1 GBM Models for Testing

To test the hypothesis, we constructed a baseline GBM model followed by experimenting on the effect of injecting different ranges of randomness on the model, and how it affects its accuracy. To eliminate any inherent order or biases in the dataset, a random shuffle was applied to the records. The shuffling process was conducted to ensure that the model's predictions would be uninfluenced by the order of data points. After shuffling the dataset, we partitioned the dataset into three distinct subsets: the training set (80% of the data) and the test set (20%). The partitioning was carried out using random sampling. We dropped the categorical data, removing any potential conversion and variability issues added by them. Following the dropping of the categorical data, we standardized individual sets using z-score normalization in relationship to the test data. We performed this standardization to facilitate model convergence and prevent features with larger scales from dominating the learning process, for each feature. Afterwards, the training of each individual GBM began.

3.1.1 Baseline GBM Model

During the training process of the baseline GBM, as shown in Figure 3, we used an early stopping technique to help determine the optimal number of trees necessary to find an efficient GBM model. We used this trained model to fit the dataset onto the test set. The best iteration was selected based on the lowest root mean squared error (RMSE) on the test set.

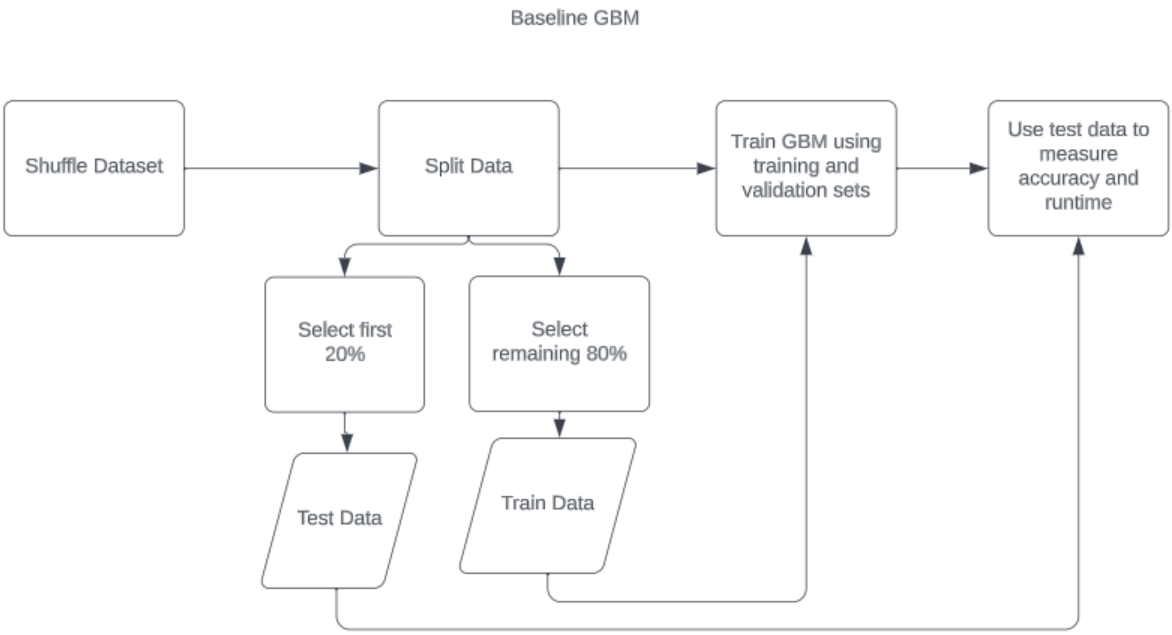

Figure 3: Flowchart representation for construction of baseline GBM

3.1.2 Noise Injected GBM

After training and validating the baseline GBM model, we used the same GBM model- process highlighted in Figure 4- avoiding any variability in models. After fitting the dataset using the trained model, we average the iterations of Gaussian noise to each produced predication set. The Gaussian noise applied upon the test set to produce the prediction set, was decided using a range of 0 to 0.1 applied on the test data.

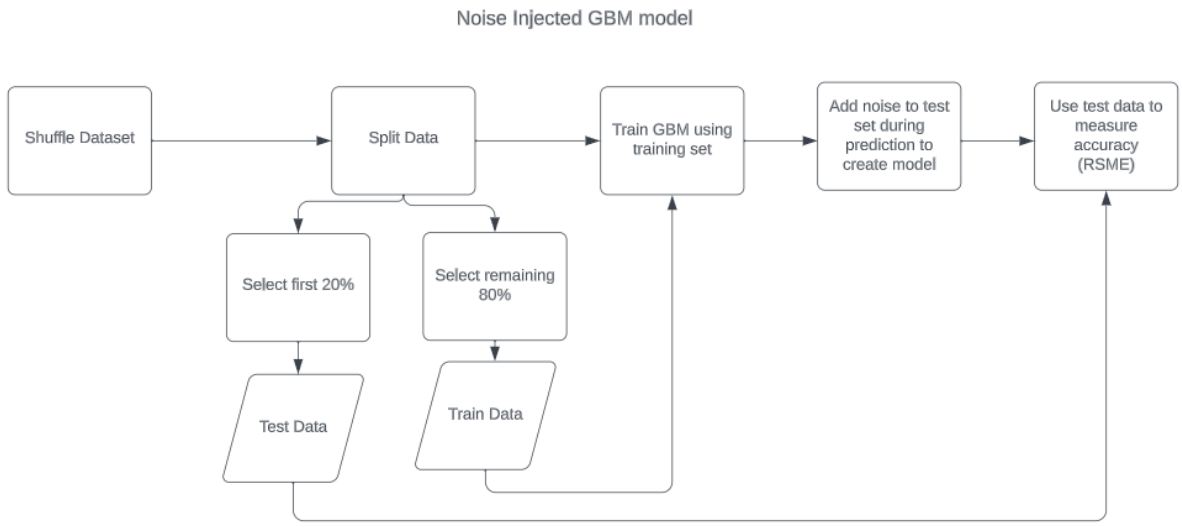

Figure 4: Flowchart representation for construction of noise injected GBM model

3.2 Testing

After constructing both the baseline GBM and the smaller GBM, we applied each model to the test data. Data collected from each model consisted of. Finally, we record and compare such data to the baseline RSME.

3.3 Creation of Synthetic Dataset

To comprehensively evaluate the impact of injected random vectors on the performance of Gradient Boosting Machines (GBMs), we generated a synthetic dataset using the following formula:

$$y = 2 \cdot x_1 + 3 \cdot x_2 + 5 \cdot x_3 + 7 \cdot x_4 + 11 \cdot x_5 + 13 \cdot x_6 + e$$

where variables (x_1, x_2, x_3, x_4, x_5, x_6, e) are all random variables uniformly distributed between 0 and 1. This formula was employed to create a synthetic dataset representative of a regression task, with the added variability introduced by the random variables.

3.3.1 Experimental Setup

To assess the impact of injected randomness on GBM models, we conducted a series of experiments using the synthetic dataset. For each experiment, we varied the standard deviation of the injected random vectors while keeping the mean at zero. The standard deviations explored were 0.1, 0.01, 0.001, and 0.0001. Additionally, we tested GBM models with varying tree sizes to observe their influence on model performance. The tree sizes considered were 1, 2, 4, 8, 16, 32, 64, 128, 256, and 512. 25 tests had been taken and averaged, within each test, to reduce outliers in the produced.

3.4 Application to Primary Dataset

To assess the feasibility and effectiveness of the observed trends on the synthetic dataset in a real-world context, we applied the injection of randomness technique to our primary dataset. The same tree sizes considered in the synthetic dataset experiments (1, 2, 4, 8, 16, 32, 64, 128, 256, and 512) were used, along with varying standard deviations of injected random vectors (0.1, 0.01, 0.001).

3.6.1 Data Preprocessing

The primary dataset underwent similar preprocessing steps as described in Section 3.1. To ensure a fair comparison, the dataset was shuffled, partitioned into training and test sets, and subjected to z-score normalization. We also dropped to the categorical data to focus on the effects of the injected vectors on a continuous dataset.

3.6.2 Experimental Setup

We conducted a series of experiments for each combination of tree size and standard deviation. The baseline GBM was trained on the preprocessed primary dataset, followed by the training of GBMs with injected random vectors. The injected random vectors followed a Gaussian distribution with the specified standard deviation and were added to the predictions.

Chapter 4: Results

In this section, we present the outcomes of our study on the effect of noise injection on gradient boosting machine regression models. Table 1 displays the results of the model on the synthetic dataset. In the table, the first column represents the number of trees present in the GBM model. The second column represents the baseline GBM model. The third, fourth, fifth, and sixth columns represent models with noise vectors injected during the prediction phase, standard deviation of the normal distribution being 0.1, 0.01, 0.001, and 0.0001 respectively. These values indicate the spread or variability of the generated random numbers (the higher the value, the higher the spread). While analyzing the data sets, we noticed a trend between increasing the value of the standard deviation of the normal distribution used to produce the noise vector and the increasing of RSME value. The following data of Table 1 is derived from the code presented in Appendix A:

Table 1: Results of noise injection on synthetic dataset

Trees	Baseline	0.1	0.01	0.001	0.0001
1	5.12898	5.13284	5.13012	5.12911	5.128979
2	4.72422	4.72985	4.72578	4.7244	4.72422
4	4.04216	4.04803	4.04338	4.04208	4.04215
8	3.02867	3.0344	3.0287	3.02856	3.0287
16	1.87931	1.88154	1.87738	1.87913	1.87936
32	0.9668	0.948	0.95996	0.96553	0.96676
64	0.55978	0.50507	0.54396	0.55734	0.55967
128	0.49918	0.43646	0.48135	0.49651	0.499083
256	0.478	0.417	0.45963	0.47531	0.47791
512	0.467	0.40616	0.44837	0.464172	0.46693

Table 2: Percentage Improvement in the RSME in relation to baseline for synthetic dataset

Trees	0.1	0.01	0.001	0.0001
1	-0.075%	-0.022%	-0.003%	0%
2	-0.119%	-0.033%	-0.004%	0%
4	-0.145%	-0.03%	0.002%	0%
8	-0.189%	-0.001%	0.004%	-0.001%
16	-0.119%	0.103%	0.01%	-0.003%
32	1.945%	0.707%	0.131%	0.004%
64	9.773%	2.826%	0.436%	0.02%
128	12.565%	3.572%	0.535%	0.019%
256	12.762%	3.843%	0.563%	0.019%
512	13.028%	3.989%	0.606%	0.015%

Upon completion of the test, we evaluated the RMSE values obtained from each model base on the percentage of RSME loss relation to that of the baseline GBM, as displayed in Table 2. Notably, the standard deviation of 0.1 exhibited an increasing more accurate outcome compared to the baseline model, higher as the number of trees increased. The standard deviations of 0.01 and 0.001 had also shown an increase in more accurate outcomes with 0.01 displaying higher than that of 0.001, but less than 0.1. Finally, a standard deviation of 0.0001 had shown little to no change in RSME value.

These findings shed light on the nuanced relationship between the magnitude of injected randomness and model performance in GBMs, providing valuable insights for further optimization and refinement of the proposed approach. This leads us to our final test into the effect our dataset with only continuous data (removing categorical), shown in Table 3.

Table 3: Noise injection on real-life dataset

Trees	Baseline	0.1	0.01	0.001	0.0001
1	18454.66032	18501.74625	18462.11948	18455.61764	18454.46248
2	17372.5802	17476.5819	17386.40235	17374.54822	17372.12841
4	15636.82659	15799.65298	15653.64272	15639.11194	15636.45145
8	13332.36948	13648.18539	13354.06861	13335.8754	13331.8357
16	11245.82086	11822.01455	11294.91744	11254.9471	11248.39744
32	10037.55543	10852.71166	10119.63984	10054.02021	10040.89564
64	9418.75334	10429.1676	9578.41437	9459.23128	9429.01942
128	8883.27492	10166.83263	9100.9054	8936.22439	8896.89951
256	8469.65882	9995.4749	8723.81745	8527.7907	8487.45078
512	8112.92608	9914.17006	8374.00577	8163.57934	8126.64751

Table 4: Percentage Improvement in the RSME in relation to baseline for the real-life dataset

Trees	0.1	0.01	0.001	0.0001
1	-0.255%	-0.04%	-0.005%	0.001%
2	-0.599%	-0.08%	-0.011%	0.003%
4	-1.041%	-0.108%	-0.015%	0.002%
8	-2.369%	-0.163%	-0.026%	0.004%
16	-5.124%	-0.437%	-0.081%	-0.023%
32	-8.121%	-0.818%	-0.164%	-0.033%
64	-10.728%	-1.695%	-0.43%	-0.109%
128	-14.449%	-2.45%	-0.596%	-0.153%
256	-18.015%	-3.001%	-0.686%	-0.21%
512	-22.202%	-3.218%	-0.624%	-0.169%

After performing the same test using the real-life dataset, analysis revealed results contrary to the earlier findings. In particular, the standard deviation of 0.1 exhibited the highest RSME gain, showcasing an increasingly accurate outcome compared to the baseline model, especially as the number of trees increased. Moreover, the standard deviations of 0.01 and 0.001 also displayed heightened RSME values, with 0.01 surpassing that of 0.001 but remaining below 0.1. Lastly, a

standard deviation of 0.0001 continued to exhibit little change in RSME value, emphasizing its consistent behavior observed in the previous analysis.

Chapter 5: Limitations

While our study has generated valuable insights into the effect of noise injection on gradient boosting machine regression models, it is essential to acknowledge certain limitations that impact the interpretation and generalization of our findings.

5.1 Compatibility of Noise Addition with Gradient Boosting Machines (GBM)

One primary limitation stems from the potential incompatibility of the noise addition approach with GBM models. It is plausible that the inherent characteristics of GBM models are not conducive to the introduction of noise, raising questions about their effectiveness within this specific modeling framework.

5.2 Handling of Categorical Attributes

Another limitation may arise from the handling of categorical attributes within our approach. The efficacy of noise addition could be influenced by the way categorical variables are processed, introducing a potential source of bias or inefficiency. Table 3 shows the importance of such categorical variables when used to create the GBM model.

Table 5: Feature importance and data type

Feature	Importance	Data Type
Horsepower	6796	Continuous
Mileage	4372	Continuous
Year	4106	Continuous
Highway fuel economy	3330	Continuous
Engine displacement	3255	Categorical
City fuel economy	3229	Continuous
Seller rating	3023	Continuous
SP ID	2396	Categorical
Latitude	2258	Continuous
Days on market	2064	Continuous
Longitude	1936	Continuous
Dealer zip	1672	Categorical
Listing ID	816	Categorical
Model name	795	Categorical
Savings amount	789	Continuous
Owner count	548	Continuous
Franchise dealer	215	Categorical
Fleet	166	Categorical
Is Cab	121	Categorical
Is New	116	Categorical
Frame damage	69	Categorical
Salvage	60	Categorical
Exterior color	56	Categorical
Has accidents	55	Categorical
Trim name	44	Categorical
Theft title	43	Categorical

5.3 Subset of Continuous Valued Variables for Noise Addition

The decision to add noise exclusively to some subset of continuous-valued variables introduces an additional layer of complexity. The effectiveness of this selective noise addition strategy may depend on the nature of the variables chosen, potentially limiting the generalizability of our approach.

5.4 Noise Addition to the Training Data

The choice to add noise to the training data exclusively may pose challenges in terms of model adaptability. This limitation raises questions about the model's ability to adapt to new data that may not have experienced the same noise during the training phase.

5.5 Dataset-Specific Features

It is conceivable that the specific characteristics of the dataset used in this study contribute to the limitations observed. Certain unique features within the data might hinder the success of our noise addition approach, emphasizing the need for careful consideration of dataset-specific nuances.

5.6 Complexity Constraints on Individual Trees

The success of our approach is contingent on limiting the complexity of individual trees within the GBM ensemble. This limitation, in addition to restricting the number of estimators, poses challenges in achieving the desired advantage from the noise addition strategy. Striking the right balance between complexity constraints and model performance remains an ongoing consideration.

In light of these limitations, future research endeavors should address these challenges and explore alternative strategies to enhance the robustness and applicability of the proposed noise addition approach. Recognizing these constraints is integral to fostering an understanding of the potential boundaries of our study's conclusions.

Chapter 6: Conclusion

In this section, we present the outcomes of our investigation into the impact of noise injection on gradient boosting machine regression models.

6.1 Trends in Results

Upon analyzing the datasets, a trend emerged concerning the standard deviation of the normal distribution used for noise vector generation and its impact on the RSME values in the real-life dataset. Notably, there is an observable increase in RSME values with an increase in the standard deviation, suggesting a correlation between the spread of the generated noise and the model's predictive accuracy.

6.2 Analysis of Synthetic Dataset Results

The experiments with injected random vectors revealed intriguing findings. Notably, a standard deviation of 0.1 led to a more accurate outcomes compared to the baseline GBM. Following a similar trend, standard deviations of 0.01 and 0.001 more accurate outcomes in relationship to that of the baseline, with that of 0.01 having more benefit than the RSME of 0.001. Finally, 0.0001 showed little to no improvement, being comparable to the RSME of the baseline. This observation suggests that a moderate level of injected randomness can have a positive impact on predictive performance in the context of continuous datasets.

6.3 Analysis of Real-World Dataset Results

The application of the randomness injection technique to our primary dataset, employing tree sizes ranging from 1 to 512 and standard deviations of 0.1, 0.01, 0.001, and 0.0001, yielded

insightful results. Unlike that of the synthetic data, stand deviations of 0.1, 0.01, and 0.001 displayed a negative effect of the RSME, with more randomness causing RSME of the prediction to increase as the size of the models increased. 0.0001 had shown a little to no decrease in RSME accuracy.

6.4 Implications and Limitations

The observed difference within our research between our synthetic dataset and real-life dataset models, prompts a cautious interpretation of our findings. The association suggests that increasing randomness, as introduced by our current noise injection method, may lead to a proportional increase or decrease in model error.

6.5 Consideration of Limitations

It is essential to acknowledge the limitations outlined in our study, including the compatibility of noise addition with gradient boosting machines, the handling of categorical attributes, and the dataset-specific features that may impact the success of our approach.

In conclusion, while our study provides valuable insights into the relationship between noise injection and model performance, the observed trends necessitate a thorough consideration of the implications and potential drawbacks of introducing randomness within the gradient boosting machine regression framework. Future research endeavors should explore alternative strategies and address the identified limitations to enhance the robustness of our approach.

REFERENCES

[1] Center of Disease Control and Prevention, "cdc.gov," Center of Disease Control and Prevention, 2023. [Online]. Available: https://gis.cdc.gov/Cancer/USCS/#/Demographics/. [Accessed 26 June 2023].

[2] "US Used cars dataset," Kaggle, 2020. [Online]. Available: https://www.kaggle.com/datasets/ananaymital/us-used-cars-dataset. [Accessed 11 June 2023].

[3] I. Encyclopædia Britannica, "Britannica," Encyclopædia Britannica, Inc, 6 May 2021. [Online]. Available: https://www.britannica.com/science/computer-science/Algorithms-and-complexity. [Accessed 8 April 2022].

[4] J. Brownlee, "Difference Between Classification and Regression in Machine Learning," 11 December 2017. [Online]. Available: https://machinelearningmastery.com/classification-versus-regression-in-machine-learning/. [Accessed 22 5 2022].

[5] Y. Su, "Prediction of air quality based on Gradient Boosting Machine Method," in *2020 International Conference on Big Data and Informatization Education (ICBDIE)*, Zhangjiajie, China, 2020.

[6] T. S. A Yovan Felix, "Flood Detection Using Gradient Boost Machine Learning Approach," in *2019 International Conference on Computational Intelligence and Knowledge Economy (ICCIKE)*, Dubai, United Arab Emirates, 2019.

[7] V. V. G. M. G. E. G. Spyros Theocharides, "Day-ahead Forecasting of Solar Power Output from Photovoltaic Systems Utilising Gradient Boosting Machines," in *2018 IEEE 7th World Conference on Photovoltaic Energy Conversion (WCPEC) (A Joint Conference of 45th IEEE PVSC, 28th PVSEC & 34th EU PVSEC)*, Waikoloa, HI, USA, 2018.

[8] W.-Y. Loh, "Regression tree models for," in *Institute of Mathematical Statistics Lecture Notes - Monograph Series*, Institute of Mathematical Statistics, 2006, pp. 210-228.

[9] P. Cunningham, "Ensemble Techniques," Academia, 2007.

[10] A. V. Konstantinov and L. V. Utkin, "A Generalized Stacking for Implementing," in *Cyber-Physical Systems*, Springer Cham, 2021, pp. 3-16.

[11] J. Xie, V. Rojkova, S. Pal and S. Coggeshall, "A Combination of Boosting and Bagging for KDD Cup 2009," in *Journal of Machine Learning Research*, San Diego, 2009.

[12] J. Brownlee, 18 August 2020. [Online]. Available: https://machinelearningmastery.com/how-to-use-statistics-to-identify-outliers-in-data/.

[13] P. Villabos, J. Sevilla, T. Besiroglu, H. Lennart, A. Ho and M. Hobbhahn, "Machine Learning Model Sizes and the Parameter Gap," 2022.

[14] J. Raymaekers, P. J. Rousseeuw, T. Verdonck and R. Yao, "Fast Linear Model Trees by PILOT," 2023.

[15] V. O. Zhilinskiy, "GLONASS Satellite Pseudorange Errors Mitigation Using Gradient Boosting Machine," in *2021 XV International Scientific-Technical Conference on Actual*

Problems Of Electronic Instrument Engineering (APEIE), Novosibirsk, Russian Federation, 2021.

[16] Y. Shi, J. Li and Z. Li, "Gradient Boosting with Piece-Wise Linear Regression Trees," in *Twenty-Eighth International Joint Conference on Artificial Intelligence*, Macao, 2019.

Appendix A

The source code for producing the models used for the continuous synthetic data can be found below:

```
 1 | import pandas as pd
 2 | import lightgbm as lgb
 3 | import numpy as np
 4 | from sklearn.model_selection import train_test_split
 5 |
 6 | SIZE = 10000
 7 | x1 = np.random.rand(SIZE)
 8 | x2 = np.random.rand(SIZE)
 9 | x3 = np.random.rand(SIZE)
10 | x4 = np.random.rand(SIZE)
11 | x5 = np.random.rand(SIZE)
12 | x6 = np.random.rand(SIZE)
13 | e = np.random.rand(SIZE)
14 |
15 | a1 = 2
16 | a2 = 3
17 | a3 = 5
18 | a4 = 7
19 | a5 = 11
20 | a6 = 13
21 | y = a1 * x1 + a2 * x2 + a3 * x3 + a4 * x4 + a5 * x5 + a6 * x6
22 | y += e
23 | data = pd.DataFrame({'y': y, 'x1': x1, 'x2': x2, 'x3': x3, 'x4': x4, 'x5': x5, 'x6': x6})
24 |
25 | trees = [1, 2, 4, 8, 16, 32, 64, 128, 256, 512]
26 | output = "Trees\tBaseline\t0.1\t\0.01\t0.001\t0.0001\n"
27 | results = []
28 | for tree_count in trees:
29 |     TRIALS = 25
30 |     errs = [0 for i in range(5)]
31 |     for i in range(TRIALS):
32 |         # Shuffle records in the dataset
33 |         data_encoded = data.sample(
34 |             frac=1, random_state=42).reset_index(drop=True)
35 |
36 |         # Splitting Dataset into train and test sets
37 |         x = data_encoded.drop(columns=['y'], axis=1)
38 |         y = data_encoded['y']
39 |
40 |         x_train, x_test, y_train, y_test = train_test_split(
41 |             x, y, test_size=0.2, random_state=42)
```

```
42 |
43 |         # Calculate mean and standard deviation for each feature in the training set
44 |         train_mean = x_train.mean()
45 |         train_std = x_train.std()
46 |
47 |         # Normalize the datasets using the training set mean and standard deviation
48 |         x_train_normalized = (x_train - train_mean) / train_std
49 |         x_test_normalized = (x_test - train_mean) / train_std
50 |
51 |         # Creating an object for the baseline GBM model with early stopping
52 |         baseline_model = lgb.LGBMRegressor(n_estimators=tree_count, random_state=42,
53 |           verbose=-1)
54 |         # Fitting the model without verbose parameter on the normalized training set
55 |         baseline_model.fit(
56 |             x_train_normalized, y_train
57 |         )
58 |         errs[0] += np.sqrt(np.mean((baseline_model.predict(
59 |             x_test_normalized) - y_test) ** 2))
60 |
61 |         # Predicting the target variable on the test set using the GBM
62 |         spreads = [0, 0.1, 0.01, 0.001, 0.0001]
63 |         for j in range(1, len(spreads)):
64 |             y_predictions = []
65 |             num_noise_vectors = 25
66 |             y_predictions.append(baseline_model.predict(x_test_normalized))
67 |             for k in range(num_noise_vectors):
68 |                 x_test_noisy = x_test_normalized + np.random.normal(
69 |                     0, spreads[j], x_test_normalized.shape)
70 |                 y_predictions.append(baseline_model.predict(x_test_noisy))
71 |
72 |             # Averaging the predictions
73 |             y_pred = np.mean(y_predictions, axis=0)
74 |
75 |             # Evaluating the performance of the GBM on the test set
76 |             errs[j] += np.sqrt(np.mean((y_pred - y_test) ** 2))
77 |     for n in range(len(errs)):
78 |         errs[n] /= TRIALS
79 |     print(tree_count, errs[0], errs[1], errs[2], errs[3], errs[4])
80 |     output += str(tree_count) + "\t" + str(errs[0]) + "\t" + str(errs[1]) + "\t" \
81 |               + str(errs[2]) + "\t" + str(
82 |         errs[3]) + "\t" + str(errs[4]) + "\n"
83 | print(output)
84 |
85 | results.append([tree_count] + errs)
86 |
87 | # Create a DataFrame from the results
```

```
88 | columns = ['Trees', 'BaseLine', '0.1', '0.01', '0.001', '0.0001']
89 | output_df = pd.DataFrame(results, columns=columns)
90 |
```

Appendix B

The source code for producing the models used for the continuous base data can be found below:

```
 1 | import pandas as pd
 2 | import lightgbm as lgb
 3 | import numpy as np
 4 | from sklearn.preprocessing import StandardScaler
 5 | from sklearn.model_selection import train_test_split
 6 |
 7 | # Reading the dataset in chunks
 8 |
 9 | chunk_size = 10000
10 | chunks = pd.read_csv(
11 |     '/used_cars_data.csv', chunksize=chunk_size, low_memory=False)
12 |
13 | # Initializing an empty list to store the chunks
14 | data_chunks = []
15 |
16 | # Iterating over the chunks and appending them to the list
17 | for chunk in chunks:
18 |     data_chunks.append(chunk)
19 |
20 | # Concatenating the chunks into a single DataFrame
21 | data = pd.concat(data_chunks, ignore_index=True)
22 |
23 | # Shuffle records in the dataset
24 | data = data.sample(frac=1, random_state=42).reset_index(drop=True)
25 |
26 | # Checking if 'Unnamed: 66' column exists before dropping it
27 | if 'Unnamed: 66' in data.columns:
28 |     data = data.drop(columns=['Unnamed: 66'], axis=1)
29 |
30 | # Checking if 'vin' column exists before dropping it
31 | if 'vin' in data.columns:
32 |     data = data.drop(columns=['vin'], axis=1)
33 |
34 | # Drop categorical columns
35 | categorical_cols = data.select_dtypes(include='object').columns
36 | data = data.drop(columns=categorical_cols)
37 |
38 | trees = [1, 2, 4, 8, 16, 32, 64, 128, 256, 512]
39 | output = "Trees\tBaseLine\t0.1\t\0.01\t0.001\t0.0001\n"
40 |
41 | results = []
```

```
42 |
43 | for tree_count in trees:
44 |     TRIALS = 25
45 |     errs = [0 for i in range(5)]
46 |     for i in range(TRIALS):
47 |         # Shuffle records in the dataset
48 |         data_encoded = data.sample(
49 |             frac=1, random_state=42).reset_index(drop=True)
50 |
51 |         # Splitting Dataset into train, valid, and test sets
52 |         x = data_encoded.drop(columns=['price'], axis=1)
53 |         y = data_encoded['price']
54 |
55 |         x_train, x_test, y_train, y_test = train_test_split(
56 |             x, y, test_size=0.2, random_state=42)
57 |
58 |         # Calculate mean and standard deviation for each feature in the training set
59 |         train_mean = x_train.mean()
60 |         train_std = x_train.std()
61 |
62 |         # Normalize the datasets using the training set mean and standard deviation
63 |         x_train_normalized = (x_train - train_mean) / train_std
64 |         x_test_normalized = (x_test - train_mean) / train_std
65 |
66 |         # Creating an object for the baseline GBM model with early stopping
67 |         baseline_model = lgb.LGBMRegressor(
68 |             n_estimators=tree_count, verbose=-1)
69 |
70 |         # Fitting the model without verbose parameter on the normalized training set
71 |         baseline_model.fit(
72 |             x_train_normalized, y_train
73 |         )
74 |         errs[0] += np.sqrt(
75 |             np.mean((baseline_model.predict(
76 |                 x_test_normalized) - y_test) ** 2))
77 |
78 |         spreads = [0, 0.1, 0.01, 0.001, 0.0001]
79 |         for j in range(1, len(spreads)):
80 |             y_predictions = []
81 |             num_noise_vectors = 25
82 |             y_predictions.append(baseline_model.predict(
83 |                 x_test_normalized))
84 |             for k in range(num_noise_vectors):
85 |                 x_test_noisy = x_test_normalized + np.random.normal(
86 |                     0, spreads[j], x_test_normalized.shape)
87 |                 y_predictions.append(baseline_model.predict(
```

```
88 |                     x_test_noisy))
89 |
90 |             # Averaging the predictions
91 |             y_pred = np.mean(y_predictions, axis=0)
92 |
93 |             # Evaluating the performance of the smaller GBM on the validation set
94 |             errs[j] += np.sqrt(np.mean((y_pred - y_test) ** 2))
95 |
96 |     for n in range(len(errs)):
97 |         errs[n] /= TRIALS
98 |     print(tree_count, errs[0], errs[1], errs[2], errs[3], errs[4])
99 |     output += str(tree_count) + "\t" + str(errs[0]) + "\t" + \
100 |               str(errs[1]) + "\t" + str(errs[2]) + "\t" + str(
101 |         errs[3]) + "\t" + str(errs[4]) + "\n"
102 | print(output)
103 |
104 | results.append([tree_count] + errs)
105 |
106 | # Create a DataFrame from the results
107 | columns = ['Trees', 'BaseLine', '0.1', '0.01', '0.001', '0.0001']
108 | output_df = pd.DataFrame(results, columns=columns)
```

ACADEMIC VITA

Hamza Bachnak

Education

The Pennsylvania State University, Harrisburg; Capital Honors College
Bachelor of Science in Computer Science **Class of 2023**
Honors Thesis: Analyzing the Effect of Noise Injection on Gradient Boosting Machine Regression Models
Relevant Coursework:

Calculus with Analytic Geometry I & II	Discrete Mathematics	
Netcentric Programming	Intro to Programming	Intermediate Programming
Database Design	Data Structures	Technical Writing
OOP with Web Design	Elemental Probability	
Computer Organization & Architecture	Machine Learning Data Science	
Software Engineering Design	Principle Programming Languages	

Skills and Certifications

Computer languages and IDE's: C++, Java, Python, Prolog, Scheme, Haskell, Visual Basic, Logisim, Ruby, JavaScript, HTML, Vue.js, React, CSS, Node.js, Angular, Next.js.

Other software and skills: Microsoft Office Products, Visual Studio, Github, Canvas, Adobe Creative Cloud.

Work Experience

Penn State Multi-Campus Research Experience for Undergraduates, The Pennsylvania State University, Harrisburg
Research Assistant **June 2023 – August 2023**

- Researched machine learning and data science topics
- Tested thesis throughout stages of programming using python
- Presented research poster to colleagues and the public
- Drafted and submitted research paper

STEM Summer Enrichment Program, The Pennsylvania State University, Harrisburg
Student **July 2017**

- Participated in various hands-on STEM activities
- Researched and presented project to large group of members and professors