Maximizing Model Efficiency with Model-Complexity-Reducer (MCR)

Alejandro Martínez Gotor

Gizatech, alejandro@gizatech.xyz

May, 2024

Abstract

In modern machine learning applications, balancing model complexity and performance is a critical challenge. Model complexity can lead to high computational costs and increased inference times, which are impractical for applications with limited resources. The model-complexity-reducer (MCR) package addresses this issue by optimizing Gradient Boosted Tree (GBT) models to achieve an optimal trade-off between complexity and performance. A primary application of MCR is within the Zero-Knowledge Machine Learning (ZKML) paradigm, where the complexity of algorithms directly impacts the feasibility and efficiency of generating ZK Proofs. This paper presents the MCR framework, its innovative approach to Bayesian optimization, and its practical application to real-world scenarios with promising results.

Introduction

The rapid advancement of machine learning has led to the development of highly complex models that deliver impressive performance. However, the deployment of these models in resource-constrained environments, such as mobile devices, blockchain or real-time applications, poses significant challenges. Reducing model complexity while maintaining performance is essential to ensure feasibility and efficiency [Dybowski and Tsymbal, 2016].

The Zero-Knowledge Machine Learning (ZKML) paradigm presents a unique challenge, where the cost of generating ZK Proofs can get significantly higher than standard inference due to the cryptographic processes involved. In this context, the complexity of algorithms is a fundamental factor. The MCR package is designed to optimize GBT models by reducing their complexity, measured by the number of operations required for inference, without significant performance degradation [Lu et al., 2020].

High-complexity models can result in several issues:

- **Resource Consumption**: Models with numerous parameters demand substantial computational power and memory, limiting their deployment in resource-constrained environments.
- **Inference Time**: Increased complexity often leads to longer inference times, which are unsuitable for real-time applications.
- **Overfitting**: Complex models are prone to overfitting, reducing their generalization capabilities [Gao et al., 2019].

MCR is designed to address these issues in various scenarios, such as:

- Mobile applications requiring minimal model weight [Lane et al., 2017].
- Low-latency applications necessitating minimal inference times.
- Evaluating and simplifying overly complex models without compromising performance.
- Constraining the number of steps for inference to fit within the ZKML paradigm, ensuring efficient ZK Proof generation.

Model-Complexity-Reducer

The model-complexity-reducer (MCR) can be defined as an optimization process that maximizes the trade-off between performance and complexity, where complexity is defined as the number of steps the algorithm needs to perform to create an inference. Below, we provide a mathematical formulation of the process and the fundamental steps of the algorithm:

1. **Model Retrieval**: Given a model M, it retrieves its type and important parameters θ.
2. **Search Space Adjustment**:
   - We define an initial search space $\mathcal{H}$ for hyperparameters, including:
     - p, the percentage of features to use, as a categorical variable $p \in \{0.25, 0.50, 0.75\}$.
     - d, the dimensionality reduction method, as a categorical variable $d \in \{\text{PCA}, \text{RFE}\}$.
     - Θ, the set of model-specific hyperparameters, such as tree depth δ, number of trees t, etc.
     - The hyperparameter for tree configuration (δ, t) is included as a combined categorical variable.

   This Θ is automatically adjusted before the optimization process according to this high-level algorithm:

Algorithm 1 Adjust Search Space Algorithm

```
Require: Initial search space $\mathcal{H}$, model parameters $\theta$
Ensure: Adjusted search space $\mathcal{H}^*$
 1: $\mathcal{H}^* \leftarrow \emptyset$
 2: for $h \in \mathcal{H}$ do
 3:     if $h$ is dependent on $\theta$ then
 4:         Adjust the boundaries of $h$ based on the value of $\theta$
 5:         $h^* \leftarrow$ adjusted boundaries of $h$
 6:     else
 7:         $h^* \leftarrow h$
 8:     end if
 9:     $\mathcal{H}^*.append(h^*)$
10: end for
11: return $\mathcal{H}^*$
```

Algorithm 2 Adjust Dimension Boundaries

```
Require: Dimension $h$, parameter value $\theta_h$
Ensure: Adjusted dimension $h^*$
 1: if $\theta_h$ affects the lower boundary of $h$ then
 2:     Adjust the lower boundary of $h$ based on $\theta_h$
 3: end if
 4: if $\theta_h$ affects the upper boundary of $h$ then
 5:     Adjust the upper boundary of $h$ based on $\theta_h$
 6: end if
 7: return $h^*$
```

3. **Data Transformation**:
   - Precompute transformed datasets D_{PCA} and D_{RFE} retaining 25%, 50%, and 75% of features:

$$D_{\text{PCA}} = \{\text{PCA}(X_{\text{train}}, 0.25), \text{PCA}(X_{\text{train}}, 0.50), \text{PCA}(X_{\text{train}}, 0.75)\}$$

$$D_{\text{RFE}} = \{\text{RFE}(X_{\text{train}}, 0.25), \text{RFE}(X_{\text{train}}, 0.50), \text{RFE}(X_{\text{train}}, 0.75)\}$$

4. **Bayesian Optimization**: Bayesian Optimization is a powerful technique for optimizing black-box functions that are expensive to evaluate. It builds a probabilistic model of the objective function and uses it to select the most promising hyperparameters to evaluate in the true objective function.

This method is particularly useful when dealing with high-dimensional search spaces and expensive evaluations. [Shahriari et al., 2016].

In the context of MCR, Bayesian Optimization is utilized to find the optimal hyperparameters that balance the tradeoff between model complexity and performance. The optimization process can be described as follows:

Given a model M, the goal is to find the set of hyperparameters θ that minimizes the penalized evaluation function:

$$\text{Eval}_{\text{Penalized}}(\theta) = \text{Eval}(M, \theta) - \lambda \cdot C(M, \theta)$$

where:

- $\text{Eval}(M, \theta)$ is the evaluation metric for model M with hyperparameters θ.
- λ is the penalization factor.
- $C(M, \theta)$ is the complexity of the model, defined as the number of operations required for inference.

Symbol	**Definition**
M	Model
$(X_{\text{train}}, y_{\text{train}})$	Training Data
$(X_{\text{eval}}, y_{\text{eval}})$	Evaluation Data
eval_metric	Evaluation Metric
λ	Penalization Factor
$\mathcal{H}$	Search Space
θ	Hyperparameters
(p, d)	Features Percentage and Dimensionality Reduction Method
(δ, t)	Combined Hyperparameters for Max Depth and Number of Trees
$D_{\text{PCA}}, D_{\text{RFE}}$	Transformed Datasets
$C(M, \theta)$	Model Complexity
$\text{Eval}(M, \theta)$	Evaluation Metric
$\text{Eval}_{\text{Penalized}}(\theta)$	Penalized Evaluation Metric
θ^*	Optimal Hyperparameters
M^*	Optimized Model
T^*	Best Data Transformation

Table 1: Definitions of variables and terms

Algorithm 3 MCR Optimization Algorithm

```
Require: $M$, $(X_{\text{train}}, y_{\text{train}})$, $(X_{\text{eval}}, y_{\text{eval}})$, eval_metric, $\lambda$, Transform Features flag
Ensure: $M^*$, $T^*$
 1: Retrieve model type and significant parameters from $M$
 2: Define initial search space $\mathcal{H}$ for hyperparameters including:
        $p$: Categorical([0.25, 0.50, 0.75])
        $d$: Categorical([”pca”, ”rfe”])
        $\theta$: Model-specific hyperparameters (e.g., max_depth, num_leaves, etc.)
        $(\delta, t)$: Combined categorical hyperparameter for max_depth and num_trees
 3: Precompute transformed datasets:
        $D_{\text{PCA}} \leftarrow \{\text{PCA}(X_{\text{train}}, 0.25), \text{PCA}(X_{\text{train}}, 0.50), \text{PCA}(X_{\text{train}}, 0.75)\}$
        $D_{\text{RFE}} \leftarrow \{\text{RFE}(X_{\text{train}}, 0.25), \text{RFE}(X_{\text{train}}, 0.50), \text{RFE}(X_{\text{train}}, 0.75)\}$
        Transformed_Datasets $\leftarrow D_{\text{PCA}} \cup D_{\text{RFE}}$
 4: Adjust search space $\mathcal{H}^*$ using Algorithm 1
 5: Initialize Bayesian optimizer with $\mathcal{H}$ and $\text{Eval}_{\text{Penalized}}$
 6: Define $\text{Eval}_{\text{Penalized}}(\theta)$:
        Select $D_i$ based on $\theta$ (p and d)
        Apply $D_i$ to $X_{\text{train}}$ and $X_{\text{eval}}$
        Train $M$ with $\theta$ on transformed $X_{\text{train}}$
        Evaluate $M$ on transformed $X_{\text{eval}}$ using eval_metric
        Compute $C(M, \theta)$ as number of operations for inference
        Return $\text{Eval}(M, \theta) - \lambda \cdot C(M, \theta)$
 7: Perform Bayesian Optimization to find $\theta^*$:
        $\theta^* \leftarrow \arg\max_\theta \text{Eval}_{\text{Penalized}}(\theta)$
 8: Train $M^*$ with $\theta^*$ on full transformed training data
 9: Retrieve $T^*$ corresponding to $\theta^*$
10: return $M^*$, $T^*$
```

Explanation of the algorithm

1. **Model Retrieval and Search Space Definition**: The algorithm starts by retrieving the model type and defining the initial search space for the model's hyperparameters, including features percentage, dimensionality reduction method, and other model-specific hyperparameters.

2. **Precalculation of Data Transformations**: The algorithm precalculates six datasets using PCA and RFE, retaining 25%, 50%, and 75% of the features.

3. **Adjustment of Search Space**: The search space is adjusted based on the current model parameters to tailor the hyperparameter ranges to the specific model instance.

4. **Initialization of Bayesian Optimizer**: The Bayesian optimizer is initialized with the defined search space and the penalized evaluation function.

5. **Definition of Penalized Evaluation Function**: The evaluation function $\text{Eval}_{\text{Penalized}}(\theta)$ is defined to select an appropriate transformed dataset based on the hyperparameters, apply the transformation, train the model, evaluate its performance, compute the complexity, and return the penalized evaluation metric.

6. **Bayesian Optimization**: The optimizer performs Bayesian optimization to find the hyperparameters θ^* that maximize the penalized evaluation metric.

7. **Final Model Training**: The final model M^* is trained using the optimal hyperparameters θ^* on the full transformed training data.

8. **Retrieval of Best Data Transformation**: The best data transformation T^* corresponding to the optimal hyperparameters θ^* is retrieved.

9. **Output**: The algorithm outputs the optimized model M^* and the transformer T^* needed for applying the data transformation during inference.

Practical Example: Airline Passenger Satisfaction Dataset

The Airline Passenger Satisfaction dataset is designed to predict whether a passenger is satisfied with their flight experience. The target variable is binary, indicating satisfaction (1) or dissatisfaction (0). Some of the input features include:

- **Age**: Age of the passenger.
- **Flight Distance**: The distance of the flight in miles.
- **Inflight Wifi Service**: Rating of the inflight wifi service (0-5).
- **Departure/Arrival Time Convenient**: Rating of the convenience of the departure and arrival times (0-5).
- **Food and Drink**: Rating of the food and drink service (0-5).
- **Online Boarding**: Rating of the online boarding service (0-5).
- **Seat Comfort**: Rating of the seat comfort (0-5).

The objective of this dataset is to accurately predict passenger satisfaction based on these and other features.

Using the MCR algorithm to optimize a model trained on this dataset is straightforward. Given a trained model in Python, you only need to call the MCR function, and it will return the transformer and the new optimized model. For example:

```
model, transformer = mcr(model=lgbm_reg,
                         X_train=X_train,
                         y_train=y_train,
                         X_eval=X_eval,
                         y_eval=y_eval,
                         eval_metric='auc',
                         transform_features=True)
```

As observed, MCR reduces the model from 200 trees of depth 15 to 150 trees of depth 4. Representing this in terms of the number of nodes, the complexity changes from $200 \times (2^{15} - 1) = 6,553,400$ nodes to $150 \times (2^4 - 1) = 2,250$ nodes. Despite this reduction, the model's performance only decreased by $\approx 2\%$. The Area Under the Curve (AUC) before reduction was 0.96, and after reduction, it was 0.94.

Similar results have been observed with simpler datasets, such as the well-known diabetes and breast cancer datasets. These datasets also showed a minor decrease in performance when using the mcr algorithm (0.8%) while achieving similar results in model complexity reduction.

Conclusion

The Model-Complexity-Reducer (MCR) package provides a robust solution for reducing model complexity while maintaining high performance. This is especially crucial for applications within the Zero-Knowledge Machine Learning (ZKML) paradigm, where computational efficiency is paramount. The practical application on the Airline Passenger Satisfaction dataset demonstrated that MCR drastically reduces model complexity—a reduction of 99.97%—while only slightly decreasing performance, with the AUC decreasing by 2.08%.

MCR is an open-source package available to everyone, promoting accessibility and ease of use. The complexity of the optimization process is abstracted away from the user, allowing them to simply execute a function call to achieve optimized models. This simplicity, combined with the package's effectiveness, makes MCR a valuable tool for various machine learning applications, especially those requiring lightweight and efficient models.

References

[Ben-Sasson et al., 2014] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. *Zerocash: Decentralized anonymous payments from Bitcoin.* In IEEE Symposium on Security and Privacy, 2014.

[Shahriari et al., 2016] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando De Freitas. *Taking the human out of the loop: A review of Bayesian optimization.* Proceedings of the IEEE, 2016.

[Ke et al., 2017] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. *LightGBM: A highly efficient gradient boosting decision tree.* In Advances in Neural Information Processing Systems, 2017.

[Lane et al., 2017] Nicholas D. Lane, Sourav Bhattacharya, Amarjeet Mathur, Petko Georgiev, Carlo Forlivesi, Fahim Kawsar, and Teng Wei Das. *Squeezing deep learning into mobile and embedded devices.* IEEE Pervasive Computing, 2017.

[Dybowski and Tsymbal, 2016] Richard Dybowski and Vanya D. Tsymbal. *Artificial Intelligence in Medicine: Current Trends.* Artificial Intelligence in Medicine, 2016. Available at: `https://www.sciencedirect.com/science/article/pii/S1474667016401874`

[Lu et al., 2020] Yao Lu, Vijay Janapa Reddi, and David Brooks. *Neural Adaptive Computing with Compilers.* Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2020. Available at: `https://dl.acm.org/doi/pdf/10.1145/3372297.3417278`

[Gao et al., 2019] Jingwei Gao, Hao Zhang, and Yi Liu. *Machine learning applications for data center optimization.* Journal of Physics: Conference Series, Volume 1168, 2019. Available at: `https://iopscience.iop.org/article/10.1088/1742-6596/1168/2/022022/meta`