COMPX521 Assignment 2

Sumeet

October 21, 2024

1 Introduction

The XGBoost algorithm is a widely used algorithm because of its accuracy and scalability as a result of gradient boosting and the scalable decision tree approach. The model improves its predictions step-by-step, leading to greater effectiveness over time [1]. However, it becomes sometimes complex and harder to interpret, which is one of the primary concerns when it comes to the decision made on the basis of a rule-based system in certain applications. As a result, it becomes essential to extract interpretable rules that can explain decisions.

Therefore, rule-based approaches are valuable when transparency and interpretability are needed in a predictive model, which is often discussed in earlier literature using the SLIPPER algorithm approach that uses the concept of a generalised version of AdaBoost, called confidence-rated boosting, to create an ensemble of rules [2].

This report will discuss the detailed analysis of the implementation of the XGBoost-Rule in the WEKA framework [5] using Java that is used to grow a single rule (considering for each boosting process) using the concept of covering algorithms by greedily optimising the objective function similar to tree-based implementation [1]. The objective is to compare the performance and size of this XGBoost-Rule implementation to the previously built tree-based model. We will analyse using various performance metrics and the number of rules generated by this approach.

2 Methodology

In this project, I adopted the similar methodology presented in [4]. The main steps of the rule-based implementation are described below.

2.1 Split Quality Optimisation

The XGBoost-Rule implemented in this project uses a greedy covering approach to optimise split quality, either on the left or right side of an attribute-value test, to grow the rule. This approach is similar to the method used in building the XGBoost decision tree. The aim of this method is to maximise the improvement

in split quality, thus the rule's effectiveness, which is directly linked to the gain in the objective function. The formula for split quality optimisation in XGBoost-Rule is as follows:

$$\frac{1}{2} \times \frac{G^2}{H + \lambda} - \gamma \times T$$

Where:

- G represents the sum of gradients at the current split point, representing the direction of the steepest descent in the loss function.

- H is the sum of second-order derivatives (Hessians), reflecting loss function curvature.

- λ and γ are regularisation terms to prevent overfitting by penalising splits with small Hessians and controlling the balance between the depth or length of the rule and the improvement in split quality. By applying more penalty to larger rule growth (T), these terms help control model complexity.

- T represents the length or depth of the rule according to the assignment, reflecting that longer rules are more complex and impose higher penalties. The formula ensures that splits with higher gain (based on gradients and Hessians) are favoured, while more complex rules are penalised, preventing overfitting.

Similar methods for minimising loss in multi-label classification have been explored in the literature [4] through gradient boosting techniques that focused on improving the objective function.

2.2 Consequent/Leaf Node Prediction

When a rule applies to an instance, XGBoost-Rule generates predictions similar to the leaf node predictions in XGBoost-Tree. For each instance that meets the rule's conditions, the prediction is calculated as:

$$-\frac{\eta \times \mathrm{G}}{\mathrm{H} + \lambda}$$

Where:

- G represents the sum of gradients for the instances falling into this leaf node and H is the sum of Hessians covered by the rule.

- λ is a regularisation term to avoid overfitting, and η is the learning rate that controls the impact of the prediction on the overall rule growth, similar to the model in a decision tree.

This method of leaf node prediction aligns with approaches that optimise the sum of gradients and Hessians with regularisation controlling overfitting as discussed in [4]. This ensures accurate predictions while also supporting effective rule growth by preventing complex rules.

2.3 Instance Not Qualifying for the Rule

The implementation also ensures that if an instance does not meet the conditions of any rule (i.e., the test condition for the rule fails), then it will return a

prediction of zero. In this way, it prevents any misclassifications by assigning arbitrary predictions to instances that do not fit into any of the defined rules.

2.4 Hyperparameters for Rule-Based Algorithm

This rule-based algorithm uses many of the same hyperparameter settings [3] as the tree-based approach. These include the learning rate (η), controlling leaf weights (λ), controlling splits (γ), minimum weight for child nodes (min_child_weight), column sampling, and subsample (the portion of samples used for each rule to avoid overfitting). This project also introduces the concept of maximum rule length in the same way as a maximum depth in an XGBoost tree. It controls how many conditions a rule can have as it grows. The default limit is 6 in this project, helping to prevent the rule from becoming too detailed and overfitting the data.

3 Experimental results

In this experiment, we compare the performance of our implemented XGBoost-Rule code against the previously implemented XGBoost-Tree on the different datasets provided. To measure accuracy, we ran 10 rounds of 10-fold cross-validation in WEKA's experimenter [5]. Both methods used the same settings (gamma = 1, subsample = 0.5), along with XGBoost's default settings [3]. The results are shown in Tables 1 and 2. Additionally, Tables 3 and 4 present the results of comparing XGBoost-Rule and XGBoost-Tree based on the number of rules and leaves generated using the 'measureNumRules' metric implemented in the code.

Table 1: Results for Regression Datasets: Root Relative Squared Error (RRSE)

Dataset	(1)	(2)	
2dplanes	23.20	22.92	•
bank8FM	31.61	33.92	◦
cpu-act	13.64	15.27	◦
cpu-small	16.06	17.31	◦
delta-ailerons	100.00	100.00	
delta-elevators	100.00	100.00	
diabetes-numeric	92.52	91.29	
kin8nm	66.98	71.79	◦
machine-cpu	37.40	36.71	
puma8NH	63.44	58.33	•
pyrim	100.23	100.23	
stock	13.13	15.70	◦
triazines	100.08	100.08	
wisconsin	111.92	114.66	

◦, • statistically significant improvement or degradation

Table 2: Results for Classification Datasets: Percent Correct

Dataset	(1)	(2)
balance-scale-weka.filter(100)	95.20	95.92
wisconsin-breast-cancer-w(100)	96.24	96.41
pima-diabetes-weka.filter(100)	73.80	75.43
ecoli-weka.filters.unsup(100)	96.34	96.07
Glass-weka.filters.unsup(100)	81.70	80.52
ionosphere-weka.filters.u(100)	92.66	92.94
iris-weka.filters.unsuper(100)	100.00	100.00
optdigits-weka.filters.un(100)	99.17	99.12
pendigits-weka.filters.un(100)	99.63	99.54
segment-weka.filters.unsu(100)	99.58	99.55
sonar-weka.filters.unsupe(100)	81.99	83.17
vehicle-weka.filters.unsu(100)	98.19	98.06
'vowel-weka.filters.unsup(100)	99.16	98.95
waveform-weka.filters.uns(100)	88.10	88.50

○, • statistically significant improvement or degradation

(1) meta.XGBoost '-S 1 -I 100 -W trees.XGBoostTree – -S 1 -colsample-bynode 1.0 -eta 0.3 -gamma 1.0 -lambda 1.0 -max-depth 6 -min-child-weight 1.0 -subsample 0.5' -7904453909409312251
(2) meta.XGBoost '-S 1 -I 100 -W rules.XGBoostRule – -S 1 -colsample-bynode 1.0 -eta 0.3 -gamma 1.0 -lambda 1.0 -max-length 6 -min-child-weight 1.0 -subsample 0.5' -7904453909409312251

Table 3: Results for Regression Datasets: measureNumRules

Dataset	(1)	(2)	
2dplanes	4708.06	100.00	•
bank8FM	133.07	100.00	•
cpu-act	4128.20	100.00	•
cpu-small	4327.26	100.00	•
delta-ailerons	100.00	100.00	
delta-elevators	100.00	100.00	
diabetes-numeric	118.14	100.00	•
kin8nm	214.28	100.00	•
machine-cpu	1911.47	100.00	•
puma8NH	4491.57	100.00	•
pyrim	100.00	100.00	
stock	645.40	100.00	•
triazines	100.00	100.00	
wisconsin	1226.33	100.00	•

○, • statistically significant improvement or degradation

Table 4: Results for Classification Datasets: measureNumRules

Dataset	(1)	(2)	
balance-scale-weka.filter(100)	254.10	100.00	•
wisconsin-breast-cancer-w(100)	186.08	100.00	•
pima-diabetes-weka.filter(100)	476.49	100.00	•
ecoli-weka.filters.unsup(100)	150.90	100.00	•
Glass-weka.filters.unsup(100)	195.46	100.00	•
ionosphere-weka.filters.u(100)	183.23	100.00	•
iris-weka.filters.unsuper(100)	110.00	100.00	•
optdigits-weka.filters.un(100)	346.21	100.00	•
pendigits-weka.filters.un(100)	361.83	100.00	•
segment-weka.filters.unsu(100)	167.51	100.00	•
sonar-weka.filters.unsupe(100)	181.19	100.00	•
vehicle-weka.filters.unsu(100)	193.26	100.00	•
'vowel-weka.filters.unsup(100)	153.06	100.00	•
waveform-weka.filters.uns(100)	1188.81	100.00	•

○, • statistically significant improvement or degradation

(1) meta.XGBoost '-S 1 -I 100 -W trees.XGBoostTree – -S 1 -colsample-bynode 1.0 -eta 0.3 -gamma 1.0 -lambda 1.0 -max-depth 6 -min-child-weight 1.0 -subsample 0.5' -7904453909409312251
(2) meta.XGBoost '-S 1 -I 100 -W rules.XGBoostRule – -S 1 -colsample-bynode 1.0 -eta 0.3 -gamma 1.0 -lambda 1.0 -max-length 6 -min-child-weight 1.0 -subsample 0.5' -7904453909409312251

4 Evaluation

The accuracy of the predictions for both the regression and classification tasks is shown above in Tables 1 and 2. For regression, the XGBoost-Rule model focuses on root relative squared error (RRSE), while for classification, it looks at overall accuracy (percent correct). When comparing the regression datasets, XGBoost-Rule and XGBoost-Tree perform similarly, but with a few notable differences. For example, there is degradation in the performance of XGBoost-Rule for *kin8nm* dataset, where XGBoost-Tree shows a statistically significant lower RRSE, likely due to its ability to better handle complex data structures. XGBoost-Tree also shows statistically significant improvements in the *cpu_act*, *bank8FM*, *cpu_small*, and *stock* datasets, though the differences are smaller. However, XGBoostRule performs slightly better on the *puma8NH* and *2dplanes* dataset, possibly due to its simpler rule-based structure, which can avoid overfitting in certain cases. For remaining datasets, minimum changes are observed.

For classification, no major improvement or degradation has been noted between implemented XGBoost-Rule when compared to XGBoost-Tree as both of them deliver almost similar results on all the datasets.

Moreover, it can be seen that the number of leaves generated using the 'measureNumRules' metric in both the rule-based and tree-based implementations is quite significant. As shown in Tables 3 and 4, XGBoost-Rule consistently generates 100 rules (leaf/consequent node) for the prediction of each classification and regression dataset, with one rule created in each boosting iteration as it ran 10 rounds of 10-fold cross-validation during the experiment. The rules

grow in a single direction based on the best split quality. On the other hand, XGBoost-Tree grows in both directions (left and right) with each split, resulting in more number of leaves with the value shown is the average number of leaves across all trees created during the boosting process for making prediction based on the complexity of the dataset when compared to rule based algorithm which is simpler and more interpretable.

5 Conclusions

To sum up, we achieved the aim of this project by successfully implemented the rule-based XGBoost algorithm and compared its performance and size with its equivalent tree-based version. Upon further examination, it is clear from the execution on the sample dataset that the rule-based algorithm is simpler and easier to interpret, as it generates a set of rules consisting of antecedents and consequents nodes by growing in single direction, while the tree-based version is more complex, taking all possible directions making it harder to interpret. Both models, however, provide almost similar performance on the data with slight differences observed on some specific data. Future improvements could include tuning hyperparameters to reduce that differences and extending the models to handle multi-class problems for broader applicability.

References

[1] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[2] William W. Cohen and Yoram Singer. A simple, fast, and effective rule learner. *Proceedings of the 16th National Conference on Artificial Intelligence*, pages 335–342, 1999.

[3] XGBoost Developers. Xgboost parameters. `https://xgboost.readthedocs.io/en/stable/parameter.html`, 2024. Accessed: 2024-10-20.

[4] Michael Rapp, Enrique L. Mencía, Johannes Fürnkranz, Vu-Linh Nguyen, and Eyke Hüllermeier. Learning gradient boosted multi-label classification rules. In *Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 124–140. Springer, 2021.

[5] Ian H. Witten, Eibe Frank, and Mark A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, Burlington, MA, 3 edition, 2011.