MLAssignment1

Ronan Li

August 2023

1 Introduction

In this study, we implemented the XGBoostTree class in Java for the Weka framework. This class was then utilized within the provided XGBoost class, aiming to facilitate to construct a comprehensive XGBoost learner in Weka.

The core principle of XGBoost is using the Taylor expansion of the loss function, specifically first and second derivative, to refine model predictions in each iteration, seeking to minimize the overall objective function. XGBoost trees incorporate regularization to reduce the risk of overfitting. The tree structures are optimized by gradient boosting framework and evaluate splits using the 'Gain'.

In this study the primary output is the algorithm module designed for integration with Weka and Java.

To evaluate the implemented XGBoost tree learning algorithm, we will compare it with the original XGBoost by performing experiments using Weka experimenter and then analyse the capabilities and potential refinements.

2 Algorithm and Implementation

In the `XGBoost` class, it is already implemented that choosing between `Logloss` and `SquaredError` based on the nominal/numeric class. First and second derivatives of the loss function are computed. Additionally, for the logistic type, a transformation using the *Sigmoid* function is performed:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

In the `XGBoostTree` class, the first and second-order derivatives, represented by g and h respectively, are received and then integrated into the tree's decision calculations.

The primary goal of boosting ensemble algorithms is to minimize a given objective function. This can be represented as:

$$\text{Obj}(\Theta) = L(\Theta) + \Omega(\Theta)$$

XGBoost uses a Taylor expansion up to the second order to handle this loss function, approximating it as:

$$\sum_i \left[L(y_i, y_{iK-1}) + g_i f_K(x_i) + \frac{1}{2} h_i f_K^2(x_i) \right]$$

In the tree, there is the following mapping relationship:

$$f_K(x) = w_{q(x)}$$

where $q(x)$ is the index of the leaf node (numbered from left to right). w represents the value of the leaf node.

The objective function can ultimately be transformed to:

$$\text{Obj} = \sum_{j=1}^{T} \left[G_j w_j + \frac{1}{2}(H_j + \lambda) w_j^2 \right] + \gamma T$$

To find the optimal w_j, we can differentiate the objective equation with respect to w_j and set it to zero:

$$\frac{\partial \text{Obj}}{\partial w_j} = G_j + (H_j + \lambda) w_j = 0$$

From which:

$$w_j^* = -\frac{G_j}{H_j + \lambda}$$

This gives us the optimal weight for a leaf.

For the Gain, it is used to measure the reduction in loss:

$$\text{Gain} = \frac{1}{2} \left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{(H_L + H_R) + \lambda} \right] - \gamma$$

In the `XGBoostTree` class specifically, the workflow can be found in the appendix.

In `XGBoost`, we also introduced a random seed and used it in `XGBoostTree` class, referred to as `seed`, to ensure consistent randomness across multiple runs.

3 Experimental Results and Discussion

3.1 Experimental Procedure and Results

In the Weka experimenter, we conducted comparative experiments with the official XGBoost using 10-fold cross-validation run 10 times across all classification and regression datasets. We then examined the classification accuracy at a significance level of 0.05 and compared the root relative squared error in both classification and regression. The results are shown in the table below:

Percent_correct Analysis for classification

Tester: `weka.experiment.PairedCorrectedTTester -G 4,5,6 -D 1 -R 2 -S 0.05 -result-matrix "weka.experiment.ResultMatrixPlainText -mean-prec 2 -stddev-prec 2 -col-name-width 0 -row-name-width 25 -mean-width 0 -stddev-width 0 -sig-width 0 -count-width 5 -print-col-names -print-row-names -enum-col-names"`

Analysing: Percent_correct
Datasets: 14
Resultsets: 2
Confidence: 0.05 (two tailed)
Sorted by: -
Date: 2023/8/25 12:14 AM

Dataset	(1) sklearn.S	(2) meta.X
balance-scale-weka.filter	95.57	73.55 *
wisconsin-breast-cancer-w	96.61	95.59
pima_diabetes-weka.filter	73.11	65.93 *
ecoli-weka.filters.unsupe	95.86	93.45 *
Glass-weka.filters.unsupe	83.02	69.61 *
ionosphere-weka.filters.u	92.99	88.46 *
iris-weka.filters.unsuper	100.00	100.00
optdigits-weka.filters.un	99.21	97.90 *
pendigits-weka.filters.un	99.68	98.60 *
segment-weka.filters.unsu	99.69	98.78 *
sonar-weka.filters.unsupe	83.85	76.24 *
vehicle-weka.filters.unsu	98.38	96.31 *
vowel-weka.filters.unsup	99.25	93.41 *
waveform-weka.filters.uns	88.63	77.83 *

Key:

(1) `sklearn.ScikitLearnClassifier '-batch 100 -learner XGBClassifier -parameters "gamma=1,subsample=0.5,colsample_bynode=0.5,tree_method= "exact " -py-command python -py-path default -server-name none' -6212485658537766441`

(2) `meta.XGBoost '-S 1 -I 100 -W trees.XGBoostTree -- -S 1 -colsample_bynode 0.5 -eta 0.3 -gamma 1.0 -lambda 1.0 -max_depth 6 -min_child_weight 1.0 -subsample 0.5' -862299050221542512`

Percent_correct Analysis for classification(only recorded differences)

Analysing: Root_relative_squared_error
Datasets: 14
Resultsets: 2

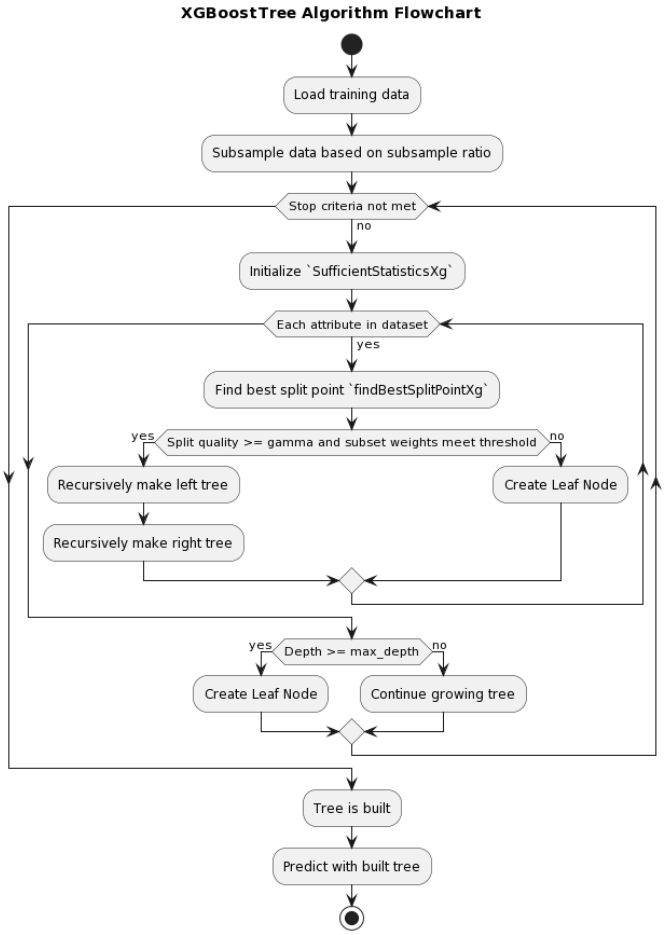
Dataset	(1) sklearn.S	(2) meta.X
2dplanes	(100) 23.24	95.64 v
bank8FM	(100) 31.10	99.99 v
cpu_act	(100) 13.74	15.71 v
cpu_small	(100) 15.81	17.48 v
delta_ailerons	(100) 100.00	100.00
delta_elevators	(100) 100.00	100.00
diabetes_numeric	(100) 90.46	95.94
kin8nm	(100) 59.83	82.81 v
machine_cpu	(100) 34.79	53.23 v
puma8NH	(100) 63.11	97.14 v
pyrim	(100) 99.81	102.26
stock	(100) 14.14	26.18 v
triazines	(100) 100.21	100.56
wisconsin	(100) 113.15	109.67
	(v/ /*)	(8/6/0)

3.2 Experimental Observations and Reflections

1. The self-implemented XGBoost is significantly slower, especially on the `2dplane` dataset. The Elapsed_Time training is 0.76 to 21.92. A probable reason could be that the my XGBoost has not optimized sorting. For every iteration, it uses `Utils.sortWithNoMissingValues` to sort all training instances anew. This hypothesis needs further validation. Given time, it might be better to employ sorting methods learned in the course to reduce the time complexity.

2. Regarding classification accuracy and root relative squared error metrics, the performance on some datasets is noticeably inferior compared to the official XGBoost, while on other datasets, it's nearly equivalent. Overall, there is a performance gap between the self-implemented XGBoost and the official one both in classification accuracy and root relative squared error.

4 Appendix A: XGBoostTree Class Flowchart


XGBoostTree Algorithm Flowchart
Load training data
Subsample data based on subsample ratio
Stop criteria not met
no
Initialize `SufficientStatisticsXg`
Each attribute in dataset
yes
Find best split point `findBestSplitPointXg`
yes
Split quality >= gamma and subset weights meet threshold
no
Recursively make left tree
Recursively make right tree
Create Leaf Node
yes
Depth >= max_depth
no
Create Leaf Node
Continue growing tree
Tree is built
Predict with built tree