XGBoost在weka的实现与分析

**引言与背景**

XGBoost的特点

Xgboost是什么？

Xgboosttree的特点是什么？与其他树区别是什么

Weka中引入算法

Weka开发机器学习算法的简介

开发的产出是要得到什么

如何实验

算法实现

Xgboost中，已经实现依据nominal/numeric class 类型来判断损失函数是Logloss还是SquaredError，和对损失函数的一阶、二阶求导。针对logic类型，还进行了Sigmoid函数转换。

代码流程图和模块图如下：

在XGBoostTree中，接收到一阶和二阶导数，可以参与到树的决策运算中。

数学原理简要介绍如下：

对于Boost集成算法，我们的目标是把一个目标函数尽量减小。

Obj(Θ)=L(Θ)+Ω(Θ)

Gradient Boosting在每次迭代中将新的树加到过往的树来进行纠错，也就使最小化损失函数L(Θ)，

Xgboost使用泰勒展开公式来操作损失函数，将损失函数近似取到二阶展开。

损失函数可以表示为：∑i[L(yi,y^K−1i)+gifK(xi)+12hif2K(xi)]

在树中有这样的映射关系：

fK(x)=wq(x)

其中q(x)叶子节点的编号（从左往右编)。w是叶子节点的取值。

也就说对于任意一个样本x,其最后会落在树的某个叶子节点上，其值为wq(x)。

目标函数最终可化为  
*Obj*=∑*j*=1*T*​[*Gj*​*wj*​+21​(*Hj*​+*λ*)*wj*2​]+*γT*

To find the optimal ��*wj*​, we can differentiate the above equation with respect to ��*wj*​ and set it to zero:

∂���∂��=��+(��+�)��=0∂*wj*​∂*Obj*​=*Gj*​+(*Hj*​+*λ*)*wj*​=0

From which:

��∗=−����+�*wj*∗​=−*Hj*​+*λGj*​​

This gives us the optimal weight for a leaf.

对于增益Gain，

Gain=12[G2LH2L+λ+G2RH2R+λ−(GL+GR)2(HL+HR)2+λ]−γ

用来衡量损失的减少。

具体在xgboosttree类中，组件图与流程如下：

在xgboost中，还引入了随机种子seed，确保随机性的一致。

随机数引入

工作原理和意义

实现细节

实验结果与讨论

在weka experimenter中对所有分类与回归的实验数据集与官方xgboost进行10次运行的10折交叉验证的对比实验后，查看了显著性水平0.05的分类准确度和分类与回归情况下的根相对平方误差结果对比。表格如下：

发现：

1. 自己实现的xgboost明显慢很多，尤其是在2dplane数据集上，训练用时对比为0.76比21.92. 猜测原因是自己实现的xgboost中，没有对排序进行优化，每次都是重新用Utils.sortWithNoMissingValues对全部训练实例排序。这里有待进一步确认，如果有时间，可以采用上课学到的排序方法降低时间复杂度。
2. 在分类准确度和根相对方差的衡量上，在一些数据集中明显差于官方xgboost，剩余数据集中基本持平。总体来说，不论是分类准确度和根相对方差，性能上是和官方xgboost有差距的。
3. 对差距较大的数据集再次实验，这次同步修改参数（eta:0.1,gamma:0,subsample:0.8,colsample\_bynode:0.8），分类任务的表现差距明显缩小。

For those datasets with large performance difference, we conducted another experiment. The parameters were adjusted in sync(eta: 0.1, gamma: 0, subsample: 0.8, colsample\_bynode: 0.8). This time the gap in performance for classification tasks was reduced.

Dataset (1) sklearn. | (2) meta.

--------------------------------------------------

pima\_diabetes-weka.filter(100) 74.92 | 74.93

--------------------------------------------------

(v/ /\*) | (0/1/0)

Conclusion

In this study, we have implemented the XGBoost algorithm within Weka and Java. The experimental results indicate several points. Firstly, our self-implemented XGBoost is notably slower in terms of training time. Secondly, although our version has variable performance across datasets, it generally lags behind the official XGBoost in both classification accuracy and root relative squared error. However, parameter adjustments did lead to reducing the gaps. We have successfully implemented the principles of XGBoost, but further work should focus on studying XGBoost source code and optimizing the algorithm.

References

Chen, T. & Guestrin, C. (2016). Xgboost: A scalable tree boosting system.

In Proceedings of the 22nd ACM SIGKDD International Conference on

Knowledge Discovery and Data Mining, KDD ’16 (p. 785–794). New York,

NY, USA: Association for Computing Machinery.

Mitchell, R. & Frank, E. (2017). Accelerating the XGBoost algorithm using GPU computing. PeerJ Computer Science, 3(e127).

<https://blog.csdn.net/qq_22238533/article/details/79477547>

<https://zhuanlan.zhihu.com/p/89572181>

<https://chat.openai.com>

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.

XGBoost, standing for eXtreme Gradient Boosting, is a widely used gradient boosting framework. Its core principle 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’.

Weka offers a platform for developing new algorithms and models. Integrating algorithms into Weka can facilitate research, provide toolkit for graphical interface. 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.

Algorithm and Implementation

In 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.

XGBoost flow and module diagrams should be illustrated here.

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.

Mathematical Principles:

Objective Function in Boosting Ensemble Algorithms:

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

*Obj*(Θ)=*L*(Θ)+Ω(Θ)

In Gradient Boosting, each iteration adds new trees to correct previous errors, essentially minimizing the loss function �(Θ)*L*(Θ)

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

∑*i*​[*L*(*yi*​,*yiK*−1​)+*gi*​*fK*​(*xi*​)+21​*hi*​*fK*2​(*xi*​)]

y goal of boosti