CHEFBOOST: A LIGHTWEIGHT BOOSTED DECISION TREE FRAMEWORK

A PREPRINT

Sefik Ilkin Serengil
Applied Data Science Department
Yapi Kredi Teknoloji
Maslak 34467 Istanbul
sefik.serengil@ykteknoloji.com.tr

October 18, 2021

ABSTRACT

Decision tree based models overwhelmingly over-perform in applied machine learning studies. In this paper, first of all a review decision tree algorithms such as ID3, C4.5, CART, CHAID, Regression Trees and some bagging and boosting methods such as Gradient Boosting, Adaboost and Random Forest have been done and then the description of the developed lightweight boosted decision tree framework - ChefBoost - has been made. Due to its widespread use and intensive choice as a machine learning programming language; Python was selected for the development of framework published also as open source package under MIT license. Moreover, the framework will build decision trees with regular if and else statements as an output. In this way, those statements can be produced and consumed programming language independently.

Keywords Machine Learning · Decision Tree · Gradient Boosting · Random Forest · Adaboost · Python

1 Introduction

Decision tree based models overwhelmingly over-perform in applied machine learning studies nowadays. The more than half of the challenge winning solutions in Kaggle are based on XGBoost in 2015. XGBoost also appears in the all top 10 solutions of the KDDCup 2015 [1]. Thereafter, LightGBM started to appear in the podium of many data science challanges since 2016 [2].

Transparency is an important indicator of decision tree adoption in applied machine learning. Similar to linear and logistic regression, we can find out the feature importance values in decision trees as well. In this way, we can join up the dots and see the big picture. However, the both linear and logistic regression are linear algorithms. Even though decision trees will not build non-linear models, they apply piece-wise linear approximation and they can solve non-linear problems. That's why, tree-based models mostly over-perform than linear/logistic regression models. Besides, it is clearly readable and understandable how a decision made with a decision tree. This functionality is missing in linear/logistic regression. Therefore, decision trees are naturally transparent, interpretable and explainable AI (xai) models.

In this paper, first of all a review decision tree algorithms have been done and then the description of the developed lightweight boosted decision tree framework - ChefBoost [1] - has been made. Due to its widespread use and intensive choice as a machine learning programming language; Python was selected for the development of framework published also as open source package under MIT license. It just depends on the Pandas [3] and NumPy [4] software packages in the background. So, it will let users to build decision trees with a few lines of code.

[1] The source code is available at https://github.com/serengil/chefboost

There are many popular core decision tree algorithms: ID3, C4.5, CART, CHAID and Regression Trees. Even though scikit-learn [5] can build decision trees simple and easy, it does not let users to choose the specific algorithm. Here, ChefBoost lets users to choose the specific decision tree algorithm.

Gradient boosting challenges many applied machine learning studies nowadays as mentioned. ChefBoost supports those boosting techniques such as Gradient Boosting and Adaboost; and also bagging methods such as Random Forest.

The data type for features is problematic subject for many existing solutions. The both LightGBM [6] and XGBoost [1] enforce users to transform categorical features to numerical. Once label encoding applied, LightGBM allows to set the type of a numerical feature to categorical in the configuration. On the other hand, XGBoost has no categorical feature support. One-hot encoding must be applied for XGBoost models and this encoding type is a costly operation. Here, ChefBoost accepts the both numerical and categorical features. In other words, it does not require to apply label or one-hot encoding in the pre-processing.

Finally, built decision trees are stored as black boxes in the existing solutions. That makes them hard to export among different systems. ChefBoost builds decision trees with regular if and else statements. In this way, those statements can be produced programming language independently. Users can build decision trees with Python and consume it in any other programming language.

2 Decision Tree Algorithms

Decision tree algorithms basically finds the most dominant feature as a decision point and then create sub trees based on sub data sets satisfying that decision. In this section, we are going to mention ID3, C4.5, CART and CHAID algorithms for classification tasks and regression trees for regression tasks.

2.1 ID3

ID3 is firstly introduced in 1986 by John Ross Quinlan [7]. It is the acronym of Iterative Dichotomiser. It sets the decision point to the most dominant feature which offers the highest information gain. The algorithm does this procedure iteratively [8].

Entropy equation is mentioned in Equation (1) where p is the probability distribution of each target class.

$$Entropy(S) = -\sum p_i . \log_2 p_i \tag{1}$$

Information gain is mentioned in Equation (2). It depends on the entropy, probability of attribute A and entropy of attribute A.

$$Gain(S, A) = Entropy(S) - \sum p(S|A).Entropy(S|A) \tag{2}$$

2.2 C4.5

ID3 comes with some disadvantages. Feature must be nominal and the data set must not include missing data. Here, C4.5 is the extended version of ID3 and it handles those cons. It was introduced by Ross Quinlan in 1993 [9]. The way the algorithm handles continuous values is to transform numerical values to greater than a threshold value or less than or equal to a threshold value. The threshold value is determined with testing all unique values as threshold.

C4.5 algorithm uses gain ratio metric to determine the most dominant feature [10]. Information gain was mentioned in section 2.1. Gain ratio is the ratio of information gain and split information as shown in Equation (3).

$$GainRatio(A) = \frac{Gain(A)}{SplitInfo(A)} \tag{3}$$

Gain ratio depends on the split information. This is a function of the probability of all possible classes for attribute A as shown in Equation (4).

$$SplitInfo(A) = -\sum \frac{|Dj|}{|D|} . \log_2 \frac{|Dj|}{|D|} \tag{4}$$

2.3 CART

CART is the acronym of classification and regression trees. It was raised in 1984 by Leo Breiman [11]. The algorithm sets the decision point to the most dominant feature which offers the lowest GINI index value [12]. Finding a gini index value of a feature A requires to calculate the probability for its each target class as shown in Equation (5).

$$Gini(A) = 1 - \sum(Pi)^2 \tag{5}$$

2.4 CHAID

CHAID was raised by Gordon V. Kass in 1980 [13]. It is the acronym of the Chi-Squared Automatic Interaction Detector. So, the algorithm sets the decision point to the most dominant feature which offers the highest chi squared value [14]. Finding the chi-squared value of a feature requires to calculate the ratio of squared observed and expected value difference to the expected value itself for each class of that feature as shown in Equation (6).

$$\chi^2 = \frac{(o-e)^2}{e} \tag{6}$$

2.5 Regression Trees

In contrast to the algorithms mentioned above, regression trees focus on the data sets with continuous target values. Decision rules will be found based on standard deviations as shown in Equation (7). The higher reducted standard deviation value, the more significance [15].

$$y = \sigma - \sum \frac{|Dj|}{|D|}\sigma_i \tag{7}$$

3 Advanced Techniques

In Section 2, we have mentioned single decision tree algorithms. The advanced bagging and boosting techniques build many decision trees to improve the scores. On the other hand, building many decision trees causes to low interpretability and explainability. Still, we are able to find out the feature importance values among many trees similar to linear / logistic regression. However, we are missing the transparent decisions of single decision trees. Luckily, some external solutions such as LIME [16] and SHAP [17] can extract how to reach a decision for complex algorithms such as deep learning and gradient boosting similar to single decision trees.

Boosting methods build a decision tree, then build another one based on the error of the previous one. In other words, they create sequential trees. Result will be the sum of the all built trees. Notice that boosting methods are developed for regression tasks. Even if you have a classification task, they transform the classification task into the regression task in the background.

On the other hand, bagging methods divide the data set into sub data sets and build many decision trees for those sub data sets meanwhile. They can be run for the both classification and regression tasks. We find the average value of the many trees in regression tasks whereas find the most frequent one in classification.

3.1 Gradient Boosting

Gradient boosting combines the terms gradient descent and boosting. It plans to build sequential decision trees based on the previous round's error. The algorithm is firstly introduced in 1999 by different researchers simultaneously [18], [19].

Suppose that we use MSE as a loss function as shown in (8). Here, y states actual values whereas y' refers to prediction.

$$J = \frac{1}{2}(y - y')^2 \tag{8}$$

Let's find the derivative of the loss function J with respect to the prediction as mentioned in (9).

$$\frac{\partial J}{\partial y'} = \frac{\partial \frac{1}{2}(y-y')^2}{\partial y'} = 2.\frac{1}{2}.(y-y').\frac{\partial(-y')}{\partial y'} = 2.\frac{1}{2}.(y-y').(-1) = y'-y \tag{9}$$

The procedure is to update predictions with the Equation (9) in the next cycle as shown in Equation (10). Here, α refers the learning rate and it is mostly picked between (0, 1].

$$y' = y' - \alpha.(\frac{\partial J}{\partial y'}) \tag{10}$$

Let's move the sign of the learning rate into the parenthesis of its multiplier as mentioned in Equation (11).

$$-\alpha.(\frac{\partial J}{\partial y'}) = -\alpha.(y'-y) = \alpha.(y-y') \tag{11}$$

This is going to be the new target value in the following cycle. In the next epoch, target values of the data set are going to be updated to the multiplication of the learning rate and the difference of its actual value and prediction in the previous epoch [20].

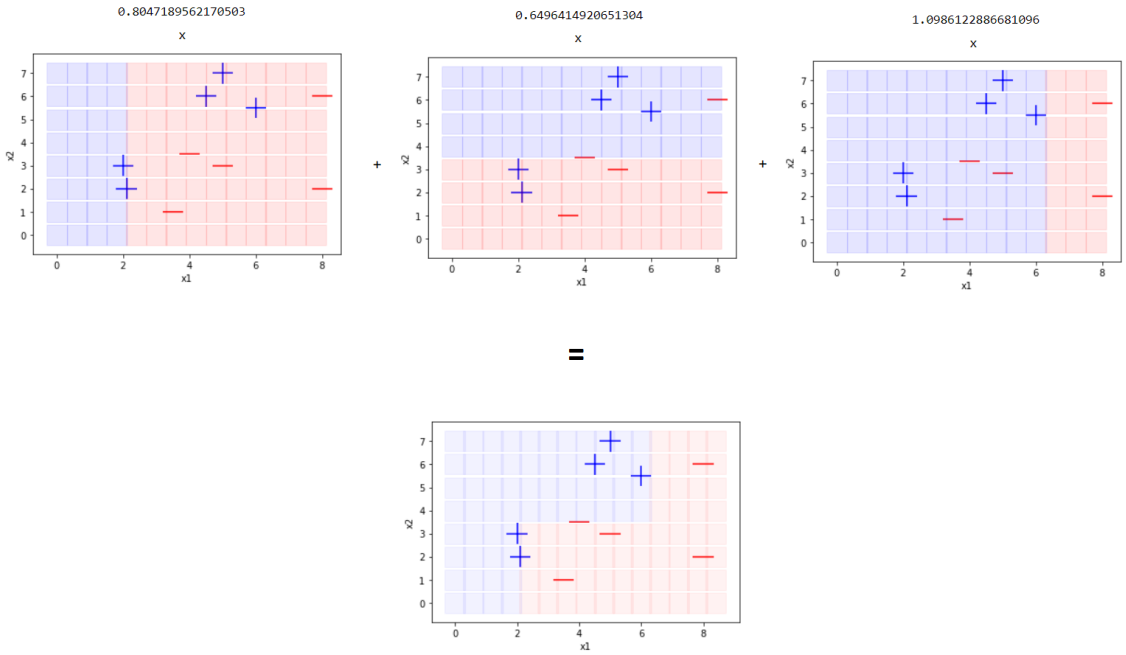
To sum up, target labels are going to be updated to the difference of the actual and prediction in the previous round to boost predictions. That's why, the data set must have continuous target values. Even if it is going to run GBM for a classification task, then the data set should be transformed to regression first [21].

3.2 Adaboost

Adaboost is the acronym of adaptive boosting. It was firstly introduced by Yoav Freund in 1997 [22]. The idea behind adaboost is that weak classifiers come together and create a strong classifier. It is expected that a weak classifier should have 50% accuracy as a prerequisite. Weak classifiers might be a single layer perceptron or a decision stump (1-level decision tree).

The weights of incorrect predictions increased whereas correct predictions decreased in each epoch as shown in Figure (1) [23]. Those weak learners cannot solve a non-linear problem individually but they can solve non-linear problems with piece-wise linear approximation when they come together and create a strong learner.

Figure 1: Adaboost Classifier



3.3 Random Forest

The whole data is fed to the single decision tree algorithms. That is because they tend to overfit. Random forest proposes to split the whole data into sub data sets randomly and build decision tree algorithms for those sub data sets. In this way, it plans to avoid overfitting. It will offers more generalized many decision trees. Random forest was firstly introduced by Tin Kam Ho in 1995 [24].

Predictions will be made with the most frequent one from those many trees for classification tasks whereas it is based on the average of those many trees in regression tasks [25]. That's why, the number of sub data sets and decision trees should be a prime number to able to make a decision.

4 Feature Importance

Feature importance is a common way to explain built models. In linear regression, the coefficients of the equation are directly related to the importance of a feature [26]. In logistic regression, coefficients are still related to the importance of a feature indirectly [27]. In decision trees, we express the importance of a feature with its found metric (e.g. entropy, gini or chi-square) times the number of instances satisfying that decision rule [28].

The both boosted trees and random forest build many decision trees. In this case, we are using the average of the feature importance value of each tree to find a global feature value in those algorithms.

5 Software Framework

We have mentioned the regular decision tree algorithms such as ID3, C4.5, CART, CHAID and Regression Trees in Section 2 and some advanced techniques such as Gradient Boosting, Adaboost and Random Forest in Section 3. All those algorithms and techniques come with a strong mathematical foundations. We plan to gather those common methods under a single roof. No software package in the open source market offers that functionality yet. In this way, decision trees can be built very easily and quickly. Besides, development of the hybrid approach enables user to switch

among algorithms in production. Due to its widespread use and intensive choice as a machine learning programming language; Python was selected for the development of the framework published also as open source package under MIT license. ChefBoost is available at Python Package Index (PyPI) [2]. Once it is installed with *pip install chefboost* command, you can import the library and access its functions under its interface.

A single decision tree building with ChefBoost for different algorithms is shown in Listing (1). Also, building bagged and boosted trees such as Gradient Boosting, Adaboost and Random Forest can be enabled with the configuration modification as shown in the code snippet. In this case, the library will build many decision trees and rules.

```
#!pip install chefboost
from chefboost import Chefboost as chef
import pandas as pd

algorithms = ["ID3", "C4.5", "CART", "CHAID", "Regression"]

df = pd.read_csv("golf.txt")
config = {'algorithm': algorithms[1]
            , 'enableGBM': False
            , 'enableRandomForest': False
            , 'enableAdaboost': False
         }

model = chef.fit(df, target_label = 'Decision', config = config)
```

Listing 1: Building a Single Decision Tree with ChefBoost

Thereafter, the library will build decision trees with regular if and else statements as shown in Listing (2) and it will store the rules in an external file named rules.py to make it re-usable.

```
#outputs/rules/rules.py
def findDecision(Outlook, Temperature, Humidity, Wind):
   if Outlook == 'Rain':
      if Wind == 'Weak':
         return 'Yes'
      elif Wind == 'Strong':
         return 'No'
      else:
         return 'No'
   elif Outlook == 'Sunny':
      if Humidity == 'High':
         return 'No'
      elif Humidity == 'Normal':
         return 'Yes'
      else:
         return 'Yes'
   elif Outlook == 'Overcast':
      return 'Yes'
   else:
      return 'Yes'
```

Listing 2: A Sample Decision Tree Output

[2]The software package with release history is available at https://pypi.org/project/chefboost/

This tree is built for the golf playing decision dataset based on outlook, temperature, humidity and wind features retrieved from the study of Quinlan [7] and shown in Table 1. Notice that feature column names in Table (1) and function input arguments in Listing (2) are same. Besides, decision labeled target column of the data set in Table (1) is mentioned in the fit command in Listing (1).

Outlook	Temperature	Humidity	Wind	Decision
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rain	Mild	High	Strong	No

Table 1: Golf Playing Behavior Dataset

Finally, you are able to find out the feature importance of the built model as shown in Listing (3). This function expects the exact path of the built model produced by the fit function.

```
chef.feature_importance("outputs/rules/rules.py")
```

Listing 3: Finding Out The Feature Importance

XGBoost [29] and LightGBM [30] are the most popular gradient boosting tools. XGBoost uses level-wise tree growth whereas LightGBM uses leaf-wise tree growth. Herein, ChefBoost uses level-wise tree growth similar to XGBoost.

Once the data set is splitted into sub data sets, random forest can build decision trees in parallel. On the other hand, single decision tree algorithms require all the data to build trees. However, once a decision node found in a single decision tree, the following decisions will be found with the sub data sets. In other words, different levels or branches can be built in parallel as well. So, ChefBoost builds branch / sub-trees in parallel for single decision trees and it builds trees in parallel for random forest.

6 Conclusion

So, we have reviewed the foundations of regular decision tree algorithms such as ID3, C4.5, CART, CHAID and Regression Trees; and and bagging and boosting methods such as Gradient Boosting, Adaboost and Random Forest. Then, we propose a lightweight framework covering those core algorithms to enable users building decision trees with just a few lines of code. In this way, we planned to build decision trees very easily and quickly. Besides, no software package in the market covers all of those algorithms yet. Also, pre-processing stages for categorical features are simplified when compared to existing software packages. Finally, the data structure of the output is designed as regular if and else statements to produce it programming language independently.

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] LightGBM Authors. Machine learning challenge winning solutions. https://github.com/Microsoft/LightGBM/blob/master/examples/README.md, 2016. [Online; accessed Oct 12, 2021].

[3] Jeff Reback, jbrockmendel, Wes McKinney, and et al. pandas-dev/pandas: Pandas 1.3.3, September 2021.

[4] Charles R Harris, K Jarrod Millman, Stéfan J van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J Smith, et al. Array programming with numpy. *Nature*, 585(7825):357–362, 2020.

[5] Olivier Grisel, Andreas Mueller, Lars, and et al. scikit-learn/scikit-learn: scikit-learn 1.0, sep 2021.

[6] 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. *Advances in neural information processing systems*, 30:3146–3154, 2017.

[7] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[8] Sefik Ilkin Serengil. A step by step id3 decision tree example. https://sefiks.com/2017/11/20/a-step-by-step-id3-decision-tree-example/, 2017. [Online; accessed Oct 12, 2021].

[9] J Ross Quinlan. *C4. 5: programs for machine learning*. Elsevier, 2014.

[10] Sefik Ilkin Serengil. A step by step c4.5 decision tree example. https://sefiks.com/2018/05/13/a-step-by-step-c4-5-decision-tree-example/, 2018. [Online; accessed Oct 12, 2021].

[11] Leo Breiman, Jerome H Friedman, Richard A Olshen, and Charles J Stone. *Classification and regression trees*. Routledge, 2017.

[12] Sefik Ilkin Serengil. A step by step cart decision tree example. https://sefiks.com/2018/08/27/a-step-by-step-cart-decision-tree-example/, 2018. [Online; accessed Oct 12, 2021].

[13] Gordon V Kass. An exploratory technique for investigating large quantities of categorical data. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 29(2):119–127, 1980.

[14] Sefik Ilkin Serengil. A step by step chaid decision tree example. https://sefiks.com/2020/03/18/a-step-by-step-chaid-decision-tree-example/, 2018. [Online; accessed Oct 12, 2021].

[15] Sefik Ilkin Serengil. A step by step regression tree example. https://sefiks.com/2018/08/28/a-step-by-step-regression-decision-tree-example/, 2018. [Online; accessed Oct 12, 2021].

[16] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should I trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 1135–1144, 2016.

[17] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4765–4774. Curran Associates, Inc., 2017.

[18] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[19] Llew Mason, Jonathan Baxter, Peter Bartlett, and Marcus Frean. Boosting algorithms as gradient descent in function space. In *Proc. NIPS*, volume 12, pages 512–518, 1999.

[20] Sefik Ilkin Serengil. A step by step gradient boosting decision tree example. https://sefiks.com/2018/10/04/a-step-by-step-gradient-boosting-decision-tree-example/, 2018. [Online; accessed Oct 12, 2021].

[21] Sefik Ilkin Serengil. A step by step gradient boosting example for classification. https://sefiks.com/2018/10/29/a-step-by-step-gradient-boosting-example-for-classification/, 2018. [Online; accessed Oct 12, 2021].

[22] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

[23] Sefik Ilkin Serengil. A step by step adaboost example. https://sefiks.com/2018/11/02/a-step-by-step-adaboost-example/, 2018. [Online; accessed Oct 12, 2021].

[24] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pages 278–282. IEEE, 1995.

[25] Sefik Ilkin Serengil. How random forests can keep you from decision tree. https://sefiks.com/2017/11/19/how-random-forests-can-keep-you-from-decision-tree/, 2017. [Online; accessed Oct 12, 2021].

[26] Sefik Ilkin Serengil. A gentle introduction to feature importance in machine learning. https://sefiks.com/2019/12/20/a-gentle-introduction-to-feature-importance-in-machine-learning/, 2019. [Online; accessed Oct 12, 2021].

[27] Sefik Ilkin Serengil. Feature importance in logistic regression for machine learning interpretability. https://sefiks.com/2021/01/06/feature-importance-in-logistic-regression/, 2021. [Online; accessed Oct 12, 2021].

[28] Sefik Ilkin Serengil. Feature importance in decision trees. https://sefiks.com/2020/04/06/feature-importance-in-decision-trees/, 2020. [Online; accessed Oct 12, 2021].

[29] Sefik Ilkin Serengil. A gentle introduction to xgboost for applied machine learning. https://sefiks.com/2019/11/03/a-gentle-introduction-to-xgboost-for-applied-machine-learning/, 2019. [Online; accessed Oct 12, 2021].

[30] Sefik Ilkin Serengil. A gentle introduction to lightgbm for applied machine learning. https://sefiks.com/2018/10/13/a-gentle-introduction-to-lightgbm-for-applied-machine-learning/, 2018. [Online; accessed Oct 12, 2021].