XGBoost: an extremely boosting method

Prepared by:
A. Kazarian, D. Senkevich, J. Grabenhofer
N. Karetnikov, R. Köfer, I. Kravchenko

WIRTSCHAFTS UNIVERSITÄT WIEN VIENNA UNIVERSITY OF ECONOMICS AND BUSINESS

EQUIS ACCREDITED AACSB ACCREDITED ASSOCIATION OF MBAS AMBA ACCREDITED

07.04.2022

Table of contents

1. Introduction

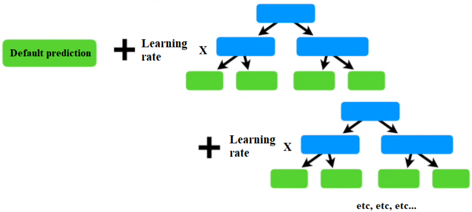
- Some **selling points** of XGBoost before we start:
  - XGboost is like generalized boosting - but EXTREME!!
  - XGboost is widely used in the winning solutions of Kaggle and KGG Cup

Original paper: Chen, T., Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. arXiv:1603.02754.

2. XGBoost Regression: Intuition

Let's follow the algorithm:

1. Calculate the residuals
2. Make the first split of residuals and calculate *Similarity score* $= \frac{(\sum \text{Residuals})^2}{\text{Number of Residuals} + \lambda}$
3. Find $Gain = Left_{Similarity} + Right_{Similarity} - Root_{Similarity}$. Leave the first split in a form which maximizes gain
4. Continue splitting. Default depth is 6 levels
5. Prune the trees: if *Gain* - γ (3 by default) < 0, we prune
6. Calculate $Output = \frac{\sum \text{Residuals}}{\text{Number of Residuals} + \lambda}$
7. Find *Predicted values* = Default (0.5 by default) + Learning rate × Output
8. Go to step 1: calculate new residuals from the new predicted values



2. XGBoost Regression: Maths behind

Loss function $L(y, p) = \frac{1}{2}(y - p)^2$
Expression to be minimized $[\sum_{i=1}^{n} L(y_i, p_i)] + \gamma T + \frac{1}{2}\lambda O^2$

In other words, $min[[\sum_{i=1}^{n} L(y_i, p^0 + O)] + \frac{1}{2}\lambda O^2]$

Using Taylor second order approx, we find that $O = \frac{-(g_1+g_2+\cdots+g_n)}{(h_1+h_2+\cdots+h_n+\lambda)}$,

where *g* is the first order deriv.: -(*y*-*p*) — residuals
and *h* is the second order deriv.: 1 x n — number of residuals

Now look at the formulas from the previous 'intuition' slide

3. XGBoost Classification: Intuition (1)

- **Let's follow the algorithm:**

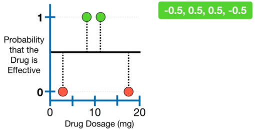
1. Calculate the residuals based on initial guess of 0.5
2. Make the first split (average of last two observations) and calculate $Similarityscore = \frac{(\sum \text{Residuals})^2}{\sum[\text{Previous probability x (1-Previous probability)}]+\lambda}$ of each leaf
3. Calculate $Gain = Left_{Similarity} + Right_{Similarity} - Root_{Similarity}$
4. Continue splitting with highest gain split. Default depth is 6 levels
5. Prune the trees by calculating Gain - γ (3 by default). If negative we prune the branch
6. Calculate $Output = \frac{\sum \text{Residuals}}{\sum[\text{Previous probability x (1-Previous probability)}]+\lambda}$
7. Find *Predicted values* = log odds of default + Learning rate (0.3 by default) × Output
8. Go to step 1: calculate new residuals from the new predicted values

Different Loss function $L(y,p) = -[y\log(p) + (1-y)\log(1-p)]$

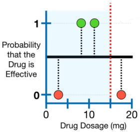
3. XGBoost Classification: Intuition (2)

WIRTSCHAFTS UNIVERSITÄT WIEN VIENNA UNIVERSITY OF ECONOMICS AND BUSINESS

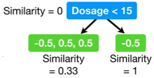
1. Calcualte residuals



2. Split

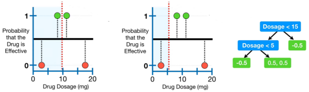


3. Calculate similarities and gain



Gain = 0.33 + 1 - 0 = 1.33

4. Repeat for other splits



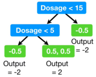
5. Pruning

Gain - γ

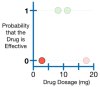
$\begin{cases} \text{If positive, then do not prune.} \\ \text{If negative, then prune.} \end{cases}$

6. Calculating output

$$\frac{(\sum \text{Residual}_i)}{\sum [\text{Previous Probability}_i \times (1 - \text{Previous Probability}_i)] + \lambda}$$



7. Predicting values



log(odds) Prediction = 0 + (0.3 x -2) = -0.6

$$\text{Probability} = \frac{e^{\text{log(odds)}}}{1 + e^{\text{log(odds)}}}$$

EQUIS ACCREDITED AACSB ACCREDITED AMBA ACCREDITED

4. Implementation of XGBoost in R

Training of the model

NOTE: XGBoost works only with numeric values

```
data = load(file = "german.rda")
dummy = dummyVars("~.", data = german[,-c(2, 5, 13, 21)])
newdata = data.frame(predict(dummy, german[,-c(2, 5, 13, 21)]))
data = cbind(newdata, german[,c(2, 5, 13, 21)])
y = recode(data$Class,'good'=1, 'bad'=0)

xgb = xgboost(data = data.matrix(data[,-c(length(data))]),
              label = y, eta = 0.3,
              max_depth = 15,
              nround=25, objective = "binary:logistic" ,
              lambda = 1,
)
```

Making predictions

```r
predict(xgb, data.matrix(data[4,-c(length(data))]))
```

```
## [1] 0.8940502
```

5. Comparison of performance

Three algorithms:

- XGBoost
- LightGBM
- CatBoost

Defining functions(1/3) – Xgboost

```
give_xgboost_AUC <- function(X_train,X_test,y_train, y_test){
xgb = xgboost(data = data.matrix(X_train[,-c(length(X_train))]),
              objective = "binary:logistic" ,
              label = y_train,nrounds=25)
vec.prob=predict(xgb, data.matrix(X_test[,-c(length(X_test))]))

pred <- prediction(vec.prob, y_test)
auc = performance(pred, measure = "auc")
return(as.numeric(auc@y.values)  )
}
```

Defining functions(2/3) – Lightgbm

```
give_lightgbm_AUC <- function(X_train,X_test,y_train, y_test){

fit <- lightgbm(data = data.matrix(X_train[,-c(length(X_train))]),
                label = y_train,
                objective = "binary",nrounds=25)

vec.prob=predict(fit, data.matrix(X_test[,-c(length(X_test))]))
pred <- prediction(vec.prob, y_test)
auc = performance(pred, measure = "auc")
return(as.numeric(auc@y.values)  )
}
```

Defining functions (3/3) – CatBoost

```r
give_catboost_AUC <- function(X_train,X_test,y_train, y_test){
train_pool <- catboost.load_pool(
  data = X_train[,-c(length(X_train))], label = y_train)
test_pool <- catboost.load_pool(
  data = X_test[,-c(length(X_test))], label = y_test)

params <- list(iterations=25, loss_function='CrossEntropy',
               eval_metric='CrossEntropy',
               use_best_model=TRUE)
fit <- catboost.train(train_pool,test_pool, params)

vec.prob= catboost.predict(fit, test_pool)
pred <- prediction(vec.prob, y_test)
auc = performance(pred, measure = "auc")
return(as.numeric(auc@y.values))
}
```

Loop for comparing the performance

```r
xgboost_AUC<-lightgbm_AUC<-catboost_AUC<-NULL
for (i in (1:1000)){
vec <- sample(dim(data)[1],700)

X_train <- data[vec,]
X_test <- data[-vec,]
y_train <- y[vec]
y_test <- y[-vec]

xgboost_AUC=c(xgboost_AUC,
              give_xgboost_AUC(X_train,X_test,y_train, y_test))
lightgbm_AUC=c(lightgbm_AUC,
               give_lightgbm_AUC(X_train,X_test,y_train, y_test))
catboost_AUC=c(catboost_AUC,
               give_catboost_AUC(X_train,X_test,y_train, y_test))}
```

Results of the comparison

```r
print(c(mean (xgboost_AUC),mean (lightgbm_AUC),mean (catboost_AUC) ))
## [1] 0.7726709 0.7728125 0.7663909
print(sum(xgboost_AUC>lightgbm_AUC))
## [1] 500
print(sum(xgboost_AUC>catboost_AUC))
## [1] 600
```

6. Distinctive features of XGBoost

When the dataset gets large, XGBoost becomes handy

- Approximate Greedy Algorithm
- Parallel Learning
- Weighted Quantile Sketch
- Sparsity-Aware Split Finding
- Cache-Aware Access
- Blocks for Out-of-Core Computation

Grand finale

Thank you for attention!

(Invite us for a coffee afterwards if you want to speak about shap values)