Predictive analytics with gradient boosting in clinical medicine

Zhongheng Zhang[1#], Yiming Zhao[1#], Aran Canes[2], Dan Steinberg[3], Olga Lyashevska[4]; written on behalf of AME Big-Data Clinical Trial Collaborative Group

[1]Department of Emergency Medicine, Sir Run Run Shaw Hospital, Zhejiang University School of Medicine, Hangzhou 310016, China; [2]Cigna, 701 Corporate Circle Drive, Raleigh, NC 27607, USA; [3]Salford Systems, Minitab, San Diego, CA, USA; [4]Galway Mayo Institute of Technology, Galway, Ireland

[#]These authors contributed equally to this work.

Correspondence to: Zhongheng Zhang. No. 3, East Qingchun Road, Hangzhou 310016, China. Email: zh_zhang1984@zju.edu.cn.

Abstract: Predictive analytics play an important role in clinical research. An accurate predictive model can help clinicians stratify risk thereby allowing the identification of a target population which might benefit from a certain intervention. Conventionally, predictive analytics is performed using parametric modeling which comes with a number of assumptions. For example, generalized linear regression models require linearity and additivity to hold for the underlying data. However, these assumptions may not hold in practice. Especially in the era of big data, a large number of covariates or features can be extracted from an electronic database which might have complex interactions and higher-order terms among the covariates. Conventional modeling methods have trouble capturing such high-dimensional relationships. However, some sophisticated machine learning techniques have been invented to handle this situation. Gradient boosting is one of these techniques which is able to recursively fit a weak learner to the residual so as to improve model performance with a gradually increasing number of iterations. It can automatically discover complex data structure, including nonlinearity and high-order interactions, even in the context of hundreds, thousands, or tens-of-thousands of potential predictors. This paper aims to introduce how gradient boosting works. The principles behind this learning machine are explained with a small example in a step-by-step manner. The formal implementation of gradient tree boosting is then illustrated with the *caret* package. In the simulated example complexity of data structure is created by generating certain interactions between the covariates. This example shows that gradient boosting can better capture these complex relationships than a generalized linear model-based approach.

Keywords: Gradient boosting; prediction; decision tree; clinical medicine

Submitted Jan 31, 2019. Accepted for publication Mar 11, 2019.
doi: 10.21037/atm.2019.03.29
View this article at: http://dx.doi.org/10.21037/atm.2019.03.29

Introduction

Predictive analytics play an important role in clinical research. If a clinical condition or outcome can be accurately predicted, interventions can be delivered to the patient population who will benefit the most. Thus, numerous data mining algorithms have been developed for clinical prediction in nearly all subspecialties. However, the most widely used method for making clinical predictions is the generalized linear model (GLM). The shortcomings of this method include, but are not limited to, linearity and additivity assumptions. In a hypothesis-driven framework, GLM methods are amiable to researchers because the trained model is relatively easy to understand and interpret.

However, with the development of electronic healthcare records, a large amount of healthcare data is readily available to clinical investigators. These often have latent complex relationships among particular variables which may not even be considered hypothetically by the investigators

developing their predictive models. Machine learning methods can be used in this setting as we can rely on the predictive algorithm to discover the nonlinearity and interaction structure in the data instead of hoping that the investigators will be wise enough to include such structure explicitly in their models.

Among the many machine learning techniques, gradient boosting is a particularly attractive approach. It was first introduced in 1999 by Stanford University Professor Jerome H. Friedman (1) and has been further developed and refined in several open source packages such as Scikit-Learn, and R. Major software developers such as SAS and IBM have developed their own implementations and Friedman's original gradient boosting machine (with refinements) has been part of the Salford Systems SPM product since 2001. As of early 2019, Google scholar reports about 10,000 citations in the scientific literature to Friedman's two papers introducing gradient boosting and applications can be found in almost all fields of scientific investigation.

In the clinical literature, gradient boosting has been successfully used to predict, among other things, cardiovascular events (2), development of sepsis (3), delirium (4) and hospital readmissions following lumbar laminectomy (5). However, the methodology is not well known among clinicians. Thus, this article aims to provide a practical guide on how to perform gradient boosting using R.

How gradient boosting works

Before we talk about the concept of gradient boosting, let's look at a simple example that reveals the main idea of formal gradient boosting.

Traditionally, modelers would try to find a function that can accurately describe the data. However, the function can only be an approximation of the data distribution and there must be errors:

$$y_i = F_1(X_i) + error_{1i} \quad [1]$$

where y_i is the outcome variable and X_i is a vector of predictors.

Suppose that the $F_1(X_i)$ function is a weak learner and the relationship between X and y is not fully described. In this situation, the error or residual is not white noise but has some correlation with y. We may want to train a second model on the error term:

$$y_i - F_1(X_i) = error_{1i} = h_1(X_i) + error_{2i} \quad [2]$$

The updated model will look something like:

$$F_2(X_i) = F_1(X_i) + h_1(X_i) \quad [3]$$

After performing this procedure iteratively we finally can fit a model at the M^{th} step:

$$F_M(X_i) = F_{M-1}(X_i) + h_{M-1}(X_i) \quad [4]$$

Gradient boosting allows one to plug in any class of weak learners $h_m(X_i)$ to improve predictive accuracy. In other words, the $h_m(X_i)$ is a weak learner that can take any functional form such as a GLM, a neural network or a decision tree. Although there is no requirement for $h_M(X_i)$ to be a specific function, it is usually a tree-based learner in practice.

The residual or error can be expressed as the loss function $L[Y, F_M(X)]$ in machine learning terminology and our objective is to build a model that minimizes this loss. More formally, the loss function is defined as the difference between the predicted $[F_M(X)]$ and the observed value (Y). The loss function can be the squared error for an ordinary least squares regression model. For loss functions that cannot be resolved with simple algebra, gradient descent is commonly employed to estimate the optimal parameters.

Gradient descent is a first-order iterative optimization algorithm for finding the minimum of a function (6). Conventionally, the gradient is calculated with respect to the model parameter. In gradient boosting, the gradient of the loss function is calculated with respect to the predicted value. That is, one takes the derivative of the loss function with respect to the predicted value. The gradient can be considered as pseudo-residual. More formally, gradient boosting proceeds with the following steps (7):

(I) Initialize the model by solving the following equation:

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma);$$ then we get:

$$F_0(x) = \frac{\sum_{i=1}^{n} y_i}{n} \quad [5]$$

(II) For m =1 to M

Compute the gradient with respect to predicted value,

$$r_{im} = -\left\{\frac{\partial L\left[y_i, F_{m-1}(x_i)\right]}{\partial F_{m-1}(x_i)}\right\}, \quad [6]$$

where *i* is the index for observations. Each observation will have one gradient value. The lowercase *m* is the number of boosting iterations with $m \in [1, M]$.

(III) Fit the weak learner $h_m(x)$ to the residuals by:

Computing the γ_m to solve the optimization problem:

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L\left[y_i, F_{m-1}(x_i) + \gamma \cdot F_m(x_i)\right] \quad [7]$$

By solving this equation we can get:

$$\gamma_m = \frac{\sum_{i=1}^{n} h_m(x_i) \cdot \left[y_i - F_{m-1}(x_i)\right]}{\sum_{i=1}^{n} h_m(x_i)^2} \quad [8]$$

(IV) Update the $F_m(x) = F_{m-1}(x) + \gamma_m \cdot h_m(x)$

An illustrative example

To make sure one really understands gradient boosting approach, we've created an example which explains the intuition behind the method.

Suppose we want to predict systolic blood pressure based on one's demographics including age, gender and obesity. Ten subjects are observed as follows:

```
> library(rpart);
> library(rpart.plot);
> dd <- data.frame(BP=c(123,111,98,154,199,
  101,91,133,116,121),
  gender=c('m','f','f','m',
'm','f','m','f','f','m'),
age=c(54,66,23,59,76,33,35,54,21,26),
obesity=c(1,0,0,1,1,0,1,0,0,0))
> dd
```

	BP	Gender	Age	obesity
1	123	M	54	1
2	111	F	66	0
3	98	F	23	0
4	154	M	59	1
5	199	M	76	1
6	101	F	33	0
7	91	M	35	1
8	133	F	54	0
9	116	F	21	0
10	121	M	26	0

The initial value for the F function is the mean. The squared error is used as the loss function, and the gradient of the loss function can be calculated as follows:

$$\begin{aligned} r_{im} &= -\left\{\frac{\partial L\left[y_i, F_{m-1}(x_i)\right]}{\partial F_{m-1}(x_i)}\right\} \\ &= -\frac{\partial\left\{\frac{1}{2}\times\left[F_{m-1}(x_i) - y_i\right]^2\right\}}{\partial F_{m-1}(x_i)} \\ &= -\frac{\partial\left\{\frac{1}{2}\times\left[F_{m-1}(x_i)^2 + y_i^2 - 2\times F_{m-1}(x_i)\cdot y_i\right]\right\}}{\partial F_{m-1}(x_i)} \\ &= y_i - F_{m-1}(x_i) \text{ for i = 1, 2, … n.} \end{aligned} \quad [9]$$

In the current iteration (m=1), the y_i is the observed blood pressure (BP) for each individual. The $F_{m-1}(x_i)$ is a vector of mean values of BP. The gradient (pseudo residual) can be computed with the following R code.

```
> dd$F0 <- mean(dd$BP)
> dd$PseudoResid1 <- dd$BP-dd$F0
```

Note that the pseudo residual considers not only the direction but also the magnitude of the difference between the observed and predicted values. Next, we will fit a regression tree to the pseudo residual:

```
> tree1 <- rpart(PseudoResid1 ~ age+gender+obesity,
  data=dd, method='anova',
  control = rpart.control(maxdepth=2,minsplit = 5))
> dd$h1 <- predict(tree1)
> dd$gamma1 <- rep(sum(dd$h1*(dd$BP-dd$F0))/
sum(dd$h1*dd$h1),10)
```

The maximum depth of the tree is set to two, making it a weak leaner. Because there are only ten observations, the minimum number of observations that must exist in a node in order for a split to be attempted is five. The prediction made by the regression tree is h1 which will be added to the $F_0(X)$ to update it to calculate $F_1(X)$:

```
> dd$F1 <- dd$F0 + dd$gamma1*dd$h1
```

Then the above process is repeated and the $F_1(X)$

function can be updated to calculate F_2 (X).

```
> dd$PseudoResid2 <- dd$BP - dd$F1
> tree2 <- rpart(PseudoResid2 ~ age+gender+obesity,
  data=dd,method='anova',
  control = rpart.control(maxdepth=2,minsplit = 5))
> dd$h2 <- predict(tree2)
> dd$gamma2<- rep(sum(dd$h2*(dd$BP-dd$F1))/sum(dd$h2*dd$h2),10);
> dd$F2 <- dd$F1+dd$gamma2*dd$h2
> dd$PseudoResid3 <- dd$BP - dd$F2
> round(dd[,c(1,5:14)],1)
```

	BP	F0	PseudoResid1	h1	gamma1	F1	PseudoResid2	h2	gamma2	F2	PseudoResid3
1	123	124.7	-1.7	3.3	1	128.0	-5.0	-1.3	1	126.7	-3.7
2	111	124.7	-13.7	30.0	1	154.7	-43.7	-19.3	1	135.3	-24.3
3	98	124.7	-26.7	-19.3	1	105.4	-7.4	-0.4	1	105.0	-7.0
4	154	124.7	29.3	30.0	1	154.7	-0.7	21.8	1	176.5	-22.5
5	199	124.7	74.3	30.0	1	154.7	44.3	21.8	1	176.5	22.5
6	101	124.7	-23.7	-19.3	1	105.4	-4.4	-0.4	1	105.0	-4.0
7	91	124.7	-33.7	-19.3	1	105.4	-14.4	-1.3	1	104.1	-13.1
8	133	124.7	8.3	3.3	1	128.0	5.0	-19.3	1	108.7	24.3
9	116	124.7	-8.7	-19.3	1	105.4	10.6	-0.4	1	105.0	11.0
10	121	124.7	-3.7	-19.3	1	105.4	15.6	-1.3	1	104.1	16.9

The above output shows how gradient boosting progresses from F0 to F2 and how the h0, pseudo residual, and F values change with fitting a regression tree to the gradient recursively. We can also visualize the regression tree with a couple of lines:

```
> par(mfrow=c(1,2))
> rpart.plot(tree1)
> rpart.plot(tree2)
```

The terminal nodes of the left tree show the percentage and mean values, which is consistent with that shown in the above table (*Figure 1*). There are five subjects (3, 6, 7, 9 and 10) in the leftmost node. These are individuals with age <45 years. The prediction of BP with mean value of the overall population is too high. Thus, the updated F1 prediction is F0 minus 19.3, which is much closer to the true value.

For gradient tree boosting, a slightly modified algorithm is employed that uses a different γ_m for each leaf node, denoted as γ_{jm} for leaf node j.

Hopefully, one now has an intuition as to how gradient boosting works. Next, we will show you how to perform gradient tree boosting in practice with a simulated dataset.

Working example

A simulated dataset was created for the purpose of illustration. The following R code generate a dataset with three categorical (Ncat =3) and four continuous (Ncont =4) variables. The outcome variable Y is a binary variable with levels of 0 and 1. Note that the linear predictor is constructed with several interaction and quadratic terms.

```
> set.seed(123);
> sampleSize=1000;
> Ncat=3; Ncont=4
> ContMean=sample(1:10,Ncont)
> ContSD=sample(1:10,Ncont)
> CatProb=runif(Ncat,min = 0.3,max = 0.7)
> for (ii in 1:Ncont) {
  assign(paste('Xcont',ii,sep = '.'),
    rnorm(sampleSize,mean = ContMean[ii],
      sd = ContSD[ii]));
}
> for (ii in 1:Ncat) {
  assign(paste('Xcat',ii,sep = '.'),
    sample(c(0,1),size = sampleSize,
      prob = c(1-CatProb[ii],CatProb[ii]),
      replace = T));
}
> linpred <- 5*Xcat.1*Xcont.1-6*Xcat.2*Xcont.2-
Xcont.3*2-Xcont.4*5*Xcat.3;
```

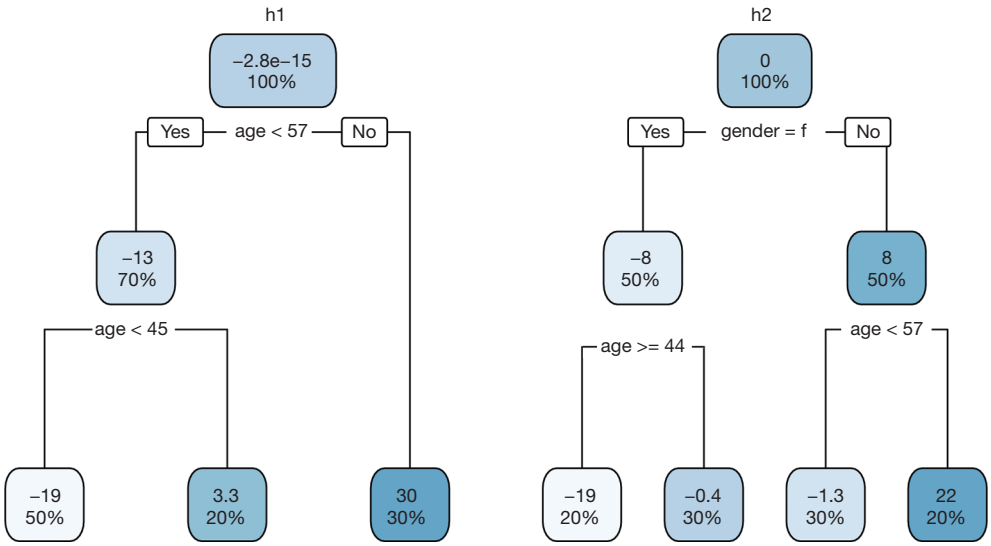

Figure 1 The predictions with h1 and h2 on the pseudo residuals. The h1 tree is split by age and there are three terminal leaf nodes. The numbers in the nodes indicate the mean value and the percentage. The h2 tree is split on age and gender resulting in 4 leaf nodes.

```
> prob <- exp(linpred)/(1 + exp(linpred))
> runis <- runif(sampleSize,0,1)
> ytest <- ifelse(runis < prob,1,0)
> df <- data.frame(Xcat.1=Xcat.1,Xcat.2=Xcat.2,
  Xcat.3=Xcat.3,Xcont.1=Xcont.1,
  Xcont.2=Xcont.2,Xcont.3=Xcont.3,
  Xcont.4=Xcont.4,Y=as.factor(ytest))
> rm(list=setdiff(ls(), "df"))
```

Split the sample into training and testing subsamples

Throughout the rest of this paper, the *caret* package (v6.0-81) will be used for training gradient tree boosting (8).

```
> library(caret)
> set.seed(123)
> index <- createDataPartition(df$Y, p=0.75, list=FALSE)
> trainSet <- df[ index,]
> testSet <- df[-index,]
```

The createDataPartition() function is used to split the dataset into training and testing subsamples. The first argument of the function is the outcome variable. Random sampling is done within the levels of Y in an attempt to balance the class distributions within the splits. The *p* argument specifies the percentage of data which goes into the training set. The function returns a matrix of row position integers corresponding to the training data.

Logistic regression model

A logistic regression model is first fit to the data to see whether the gradient boosting model is able to improve predictive performance when there are arbitrary interactions and non-linear terms among the covariates.

```
> xnam <- c(paste("Xcat", 1:3,sep='.'),paste("Xcont", 1:4,sep='.'))
> Logitformula <- as.formula(paste("Y ~ ", paste(xnam,
collapse= "+")))
> LogitMod <- glm(Logitformula,trainSet,family = 'binomial')
> logitPred <- predict(LogitMod,newdata = testSet,type =
'response')
> library(pROC)
> LogitROC <- roc(testSet$Y,logitPred)
> auc(LogitROC)
Area under the curve: 0.9218
> ci.auc(LogitROC)
95% CI: 0.8748-0.9687 (DeLong)
```

The output shows that the area under the receiver

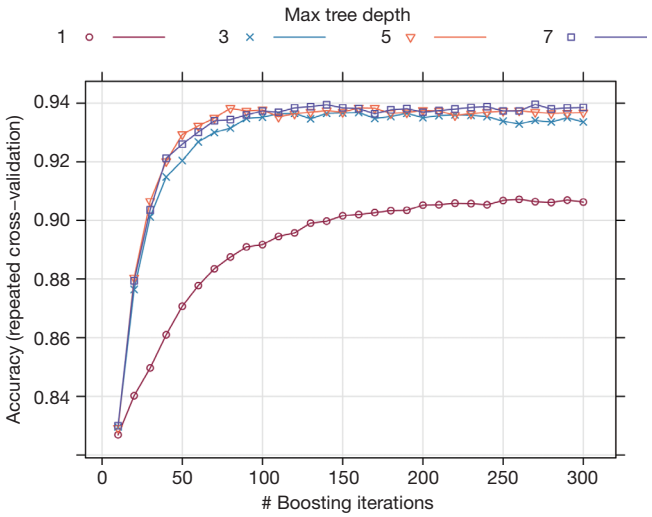

Figure 2 The gradient boosting training process. The horizontal axis shows the number of boosting iterations (number of trees). The vertical axis shows the accuracy. The maximum tree depth takes a value of 1, 3, 5 or 7.

operating characteristic curve (AUROC) of the logistic regression model is 0.92 (95% CI: 0.87–0.97).

Gradient tree boosting

```
> fitControl <- trainControl(## 10-fold CV
  method = "repeatedcv",
  number = 10, ## repeated ten times
  repeats = 10)
```

The trainControl() function controls many computational nuances of the train() function. In the example, ten-fold cross validation is carried out. Specifically, the whole data sample is randomly divided into ten blocks of roughly equal size. Each of the blocks is left out in turn and the other nine blocks are used to train the model. The held-out block is predicted on and these predictions are summarized into some type of performance measure such as accuracy for classification, and sum of squared error for regression. The above procedure is repeated ten times, resulting in 10×10=100 estimates of performance. These are then averaged to get the overall resampled estimate.

```
> gbmGrid <- expand.grid(interaction.depth = c(1,3,5,7),
  n.trees = (1:30)*10,
  shrinkage = 0.1,
  n.minobsinnode = 50)
```

The expand.grid() function creates a data frame for all combinations of the supplied vectors or factors. Here we need to introduce a new term called a hyperparameter. The hyperparameters define how our model is actually structured. They are different from model parameters and cannot be trained from the data. Grid search is arguably the most basic hyperparameter tuning method. With this scheme, we simply build a model for each possible combination of all of the hyperparameter values provided, evaluate each model and select the architecture which produces the best results. The hyperparameters of decision tree modeling include the maximum depth of each tree (*interaction.depth*), number of trees (*n.trees*), learning rate (*shrinkage*) and the minimum number of observations in the terminal nodes of the trees (*n.minobsinnode*).

```
> gbmMod <- train(Logitformula,
  trainSet, method='gbm', verbose = FALSE,
  trControl = fitControl,
  tuneGrid = gbmGrid)
```

Finally, the model can be trained with the train() function. The training process can be viewed with the generic plot() function.

```
> trellis.par.set(caretTheme())
> plot(gbmMod);
```

Figure 2 shows the training process with one grid searching strategy. The results indicate that the accuracy increases rapidly from iteration number 1 to 100, but then levels off after 100 iterations. A tree depth of more than three will not continue to improve predictive accuracy.

```
> gbmPred <- predict(gbmMod,newdata=testSet,
  type='prob')
> gbmROC <- roc(testSet[,'Y'],gbmPred[,2])
> auc(gbmROC)
Area under the curve: 0.9842
> ci.auc(gbmROC)
95% CI: 0.9715-0.9969 (DeLong)
> roc.test(gbmROC, LogitROC)

  DeLong's test for two correlated ROC
```

```
Curves

data: gbmROC and LogitROC
Z = 2.6603, p-value = 0.007807
alternative hypothesis: true difference in AUC is not equal to 0
sample estimates:
AUC of roc1 AUC of roc2
0.9841951 0.9217656
```

The result shows that AUROC of the gradient boosting model (0.98; 95% CI: 0.972–0.997) is significantly higher than the logistic regression model (with a P value of 0.008 using DeLong's test).

Conclusions

Traditional statistical modeling has long been plagued by the assumption that the particular functional form of the relationship between the outcome and predictor variables has been accurately captured by the modeler. New data mining techniques, such as gradient boosting, allow algorithms to detect potentially latent variables such as interaction and higher order terms which are difficult to explicitly model.

This paper has illustrated a step-by-step approach to gradient boosting and shown that it can lead to higher predictive power in a simulated dataset than logistic regression. The authors hope that this paper will be useful to clinicians and other health care researchers who want to apply gradient boosting to their own data sets with the expectation of an increase in predictive power.

Acknowledgements

None.

Cite this article as: Zhang Z, Zhao Y, Canes A, Steinberg D, Lyashevska O; written on behalf of AME Big-Data Clinical Trial Collaborative Group. Predictive analytics with gradient boosting in clinical medicine. Ann Transl Med 2019;7(7):152. doi: 10.21037/atm.2019.03.29

Footnote

Conflicts of Interest: The authors have no conflicts of interest to declare.

References

1. Friedman JH. Greedy function approximation: A gradient boosting machine. The Annals of Statistics 2001;29:1189-232.
2. Zhao J, Feng Q, Wu P, et al. Learning from Longitudinal Data in Electronic Health Record and Genetic Data to Improve Cardiovascular Event Prediction. Sci Rep 2019;9:717.
3. Delahanty RJ, Alvarez J, Flynn LM, et al. Development and Evaluation of a Machine Learning Model for the Early Identification of Patients at Risk for Sepsis. Ann Emerg Med 2019;73:334-44.
4. Wong A, Young AT, Liang AS, et al. Development and Validation of an Electronic Health Record-Based Machine Learning Model to Estimate Delirium Risk in Newly Hospitalized Patients Without Known Cognitive Impairment. JAMA Netw Open 2018;1:e181018.
5. Kalagara S, Eltorai AEM, Durand WM, et al. Machine learning modeling for predicting hospital readmission following lumbar laminectomy. J Neurosurg Spine 2018;30:344-52.
6. Ruder S. An overview of gradient descent optimization algorithms. Vol. cs.LG, arXiv.org. 2016.
7. Friedman JH, Meulman JJ. Multiple additive regression trees with application in epidemiology. Stat Med 2003;22:1365-81.
8. Kuhn M. caret: Classification and Regression Training. Available online: https://CRAN.R-project.org/package=caret. 2017.

