Gradient boosting

From Wikipedia, the free encyclopedia

Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function.

The idea of gradient boosting originated in the observation by Leo Breiman [1] that boosting can be interpreted as an optimization algorithm on a suitable cost function. Explicit regression gradient boosting algorithms were subsequently developed by Jerome H. Friedman[2][3] simultaneously with the more general functional gradient boosting perspective of Llew Mason, Jonathan Baxter, Peter Bartlett and Marcus Frean .[4][5] The latter two papers introduced the abstract view of boosting algorithms as iterative functional gradient descent algorithms. That is, algorithms that optimize a cost functional over function space by iteratively choosing a function (weak hypothesis) that points in the negative gradient direction. This functional gradient view of boosting has led to the development of boosting algorithms in many areas of machine learning and statistics beyond regression and classification.

Contents

Informal introduction

(This section follows the exposition of gradient boosting by Li.[6])

Like other boosting methods, gradient boosting combines weak learners into a single strong learner, in an iterative fashion. It is easiest to explain in the least-squares regression setting, where the goal is to learn a model F that predicts values $\hat{y} = F(x)$, minimizing the mean squared error $(\hat{y} - y)^2$ to the true values y (averaged over some training set).

At each stage $1 \leq m \leq M$ of gradient boosting, it may be assumed that there is some imperfect model F_m (at the outset, a very weak model that just predicts the mean y in the training set could be used). The gradient boosting algorithm does not change F_m in any way; instead, it improves on it by constructing a new model that adds an estimator h to provide a better model $F_{m+1}(x) = F_m(x) + h(x)$. The question is now, how to find h? The gradient boosting solution starts with the observation that a perfect h would imply

$$F_{m+1} = F_m(x) + h(x) = y$$

or, equivalently,

$$h(x) = y - F_m(x).$$

Therefore, gradient boosting will fit h to the residual $y - F_m(x)$. Like in other boosting variants, each F_{m+1} learns to correct its predecessor F_m. A generalization of this idea to other loss functions than squared error (and to classification and ranking problems) follows from the observation that residuals $y - F(x)$ are the negative gradients of the squared error loss function $\frac{1}{2}(y - F(x))^2$. So, gradient boosting is a gradient descent algorithm; and generalizing it entails "plugging in" a different loss and its gradient.

Algorithm

In many supervised learning problems one has an output variable y and a vector of input variables x connected together via a joint probability distribution $P(x, y)$. Using a training set $\{(x_1, y_1), \dots, (x_n, y_n)\}$ of known values of x and corresponding values of y, the goal is to find an approximation $\hat{F}(x)$ to a function $F^*(x)$ that minimizes the expected value of some specified loss function $L(y, F(x))$:

$$F^* = \arg\min_F \mathbb{E}_{x,y}[L(y, F(x))].$$

Gradient boosting method assumes a real-valued y and seeks an approximation $\hat{F}(x)$ in the form of a weighted sum of functions $h_i(x)$ from some class $\mathcal{H}$, called base (or weak) learners:

$$F(x) = \sum_{i=1}^{M} \gamma_i h_i(x) + \text{const}.$$

In accordance with the empirical risk minimization principle, the method tries to find an approximation $\hat{F}(x)$ that minimizes the average value of the loss function on the training set. It does so by starting with a model, consisting of a constant function $F_0(x)$, and incrementally expanding it in a greedy fashion:

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

$$F_m(x) = F_{m-1}(x) + \arg\min_{f \in \mathcal{H}} \sum_{i=1}^{n} L(y_i, F_{m-1}(x_i) + f(x_i)),$$

where f is restricted to be a function from the class $\mathcal{H}$ of base learner functions.

However, the problem of choosing at each step the best f for an arbitrary loss function L is a hard optimization problem in general, and so we'll "cheat" by solving a much easier problem instead.

The idea is to apply a steepest descent step to this minimization problem. If we only cared about predictions at the points of the training set, and f were unrestricted, we'd update the model per the following equation, where we view $L(y, f)$ not as a functional of f, but as a function of a vector of values $f(x_1), \dots, f(x_n)$:

$$F_m(x) = F_{m-1}(x) - \gamma_m \sum_{i=1}^{n} \nabla_f L(y_i, F_{m-1}(x_i)),$$

$$\gamma_m = \underset{\gamma}{\arg\min} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) - \gamma \frac{\partial L(y_i, F_{m-1}(x_i))}{\partial f(x_i)}\right).$$

But as f must come from a restricted class of functions (that's what allows us to generalize), we'll just choose the one that most closely approximates the gradient of L. Having chosen f, the multiplier γ is then selected using line search just as shown in the second equation above.

In pseudocode, the generic gradient boosting method is:[2][7]

```
Input: training set $\{(x_i, y_i)\}_{i=1}^{n}$, a differentiable loss function $L(y, F(x))$, number of iterations $M$.

Algorithm:

  1. Initialize model with a constant value:
         $F_0(x) = \underset{\gamma}{\arg\min} \sum_{i=1}^{n} L(y_i, \gamma).$
  2. For $m$ = 1 to $M$:
      1. Compute so-called pseudo-residuals:
             $r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \ldots, n.$
      2. Fit a base learner $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^{n}$.
      3. Compute multiplier $\gamma_m$ by solving the following one-dimensional optimization problem:
             $\gamma_m = \underset{\gamma}{\arg\min} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right).$
      4. Update the model:
             $F_m(x) = F_{m-1}(x) + \gamma_m h_m(x).$
  3. Output $F_M(x)$.
```

Gradient tree boosting

Gradient boosting is typically used with decision trees (especially CART trees) of a fixed size as base learners. For this special case Friedman proposes a modification to gradient boosting method which improves the quality of fit of each base learner.

Generic gradient boosting at the m-th step would fit a decision tree $h_m(x)$ to pseudo-residuals. Let J be the number of its leaves. The tree partitions the input space into J disjoint regions $R_{1m}, \ldots, R_{Jm}$ and predicts a constant value in each region. Using the indicator notation, the output of $h_m(x)$ for input x can be written as the sum:

$$h_m(x) = \sum_{j=1}^{J} b_{jm} I(x \in R_{jm}),$$

where b_{jm} is the value predicted in the region R_{jm}.[8]

Then the coefficients b_{jm} are multiplied by some value γ_m, chosen using line search so as to minimize the loss function, and the model is updated as follows:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x), \quad \gamma_m = \underset{\gamma}{\arg\min} \sum_{i=1}^{n} L(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)).$$

Friedman proposes to modify this algorithm so that it chooses a separate optimal value γ_{jm} for each of the tree's regions, instead of a single γ_m for the whole tree. He calls the modified algorithm "TreeBoost". The coefficients b_{jm} from the tree-fitting procedure can be then simply discarded and the model update rule becomes:

$$F_m(x) = F_{m-1}(x) + \sum_{j=1}^{J} \gamma_{jm} h_m(x) I(x \in R_{jm}), \quad \gamma_{jm} = \underset{\gamma}{\arg\min} \sum_{x_i \in R_{jm}} L(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)).$$

Size of trees

J, the number of terminal nodes in trees, is the method's parameter which can be adjusted for a data set at hand. It controls the maximum allowed level of interaction between variables in the model. With $J = 2$ (decision stumps), no interaction between variables is allowed. With $J = 3$ the model may include effects of the interaction between up to two variables, and so on.

Hastie et al.[7] comment that typically $4 \leq J \leq 8$ work well for boosting and results are fairly insensitive to the choice of J in this range, $J = 2$ is insufficient for many applications, and $J > 10$ is unlikely to be required.

Regularization

Fitting the training set too closely can lead to degradation of the model's generalization ability. Several so-called regularization techniques reduce this overfitting effect by constraining the fitting procedure.

One natural regularization parameter is the number of gradient boosting iterations M (i.e. the number of trees in the model when the base learner is a decision tree). Increasing M reduces the error on training set, but setting it too high may lead to overfitting. An optimal value of M is often selected by monitoring prediction error on a separate validation data set. Besides controlling M, several other regularization techniques are used.

Shrinkage

An important part of gradient boosting method is regularization by shrinkage which consists in modifying the update rule as follows:

$$F_m(x) = F_{m-1}(x) + \nu \cdot \gamma_m h_m(x), \quad 0 < \nu \leq 1,$$

where parameter ν is called the "learning rate".

Empirically it has been found that using small learning rates (such as $\nu < 0.1$) yields dramatic improvements in model's generalization ability over gradient boosting without shrinking ($\nu = 1$).[7] However, it comes at the price of increasing computational time both during training and querying: lower learning rate requires more iterations.

Stochastic gradient boosting

Soon after the introduction of gradient boosting Friedman proposed a minor modification to the algorithm, motivated by Breiman's bagging method.[3] Specifically, he proposed that at each iteration of the algorithm, a base learner should be fit on a subsample of the training set drawn at random without replacement.[9] Friedman observed a substantial improvement in gradient boosting's accuracy with this modification.

Subsample size is some constant fraction f of the size of the training set. When f = 1, the algorithm is deterministic and identical to the one described above. Smaller values of f introduce randomness into the algorithm and help prevent overfitting, acting as a kind of regularization. The algorithm also becomes faster, because regression trees have to be fit to smaller datasets at each iteration. Friedman[3] obtained that $0.5 \leq f \leq 0.8$ leads to good results for small and moderate sized training sets. Therefore, f is typically set to 0.5, meaning that one half of the training set is used to build each base learner.

Also, like in bagging, subsampling allows one to define an out-of-bag error of the prediction performance improvement by evaluating predictions on those observations which were not used in the building of the next base learner. Out-of-bag estimates help avoid the need for an independent validation dataset, but often underestimate actual performance improvement and the optimal number of iterations.[10]

Number of observations in leaves

Gradient tree boosting implementations often also use regularization by limiting the minimum number of observations in trees' terminal nodes (this parameter is called n.minobsinnode in the R gbm package[10]). It is used in the tree building process by ignoring any splits that lead to nodes containing fewer than this number of training set instances.

Imposing this limit helps to reduce variance in predictions at leaves.

Penalize Complexity of Tree

Another useful regularization techniques for gradient boosted trees is to penalize model complexity of the learned model. [11] The model complexity can be defined proportional number of leaves in the learned trees. The jointly optimization of loss and model complexity corresponds to a post-pruning algorithm to remove branches that fail to reduce the loss by a threshold. Other kinds of regularization such as l2 penalty on the leave values can also be added to avoid overfitting.

Usage

Recently, gradient boosting has gained some popularity in the field of learning to rank. The commercial web search engines Yahoo[12] and Yandex[13] use variants of gradient boosting in their machine-learned ranking engines.

Names

The method goes by a variety of names. Friedman introduced his regression technique as a "Gradient Boosting Machine" (GBM).[2] Mason, Baxter et. el. described the generalized abstract class of algorithms as "functional gradient boosting".[4][5]

A popular open-source implementation[10] for R calls it "Generalized Boosting Model". Commercial implementations from Salford Systems use the names "Multiple Additive Regression Trees" (MART) and TreeNet, both trademarked.

See also

- AdaBoost
- Random forest

References

1. Brieman, L. "Arcing The Edge (http://statistics.berkeley.edu/sites/default/files/tech-reports/486.pdf)" (June 1997)
2. Friedman, J. H. "Greedy Function Approximation: A Gradient Boosting Machine. (http://www-stat.stanford.edu/~jhf/ftp/trebst.pdf)" (February 1999)
3. Friedman, J. H. "Stochastic Gradient Boosting. (https://statweb.stanford.edu/~jhf/ftp/stobst.pdf)" (March 1999)
4. Mason, L.; Baxter, J.; Bartlett, P. L.; Frean, Marcus (1999). "Boosting Algorithms as Gradient Descent" (PDF). In S.A. Solla and T.K. Leen and K. Müller. Advances in Neural Information Processing Systems 12. MIT Press. pp. 512–518.
5. Mason, L.; Baxter, J.; Bartlett, P. L.; Frean, Marcus (May 1999). Boosting Algorithms as Gradient Descent in Function Space (PDF).
6. Cheng Li. "A Gentle Introduction to Gradient Boosting" (PDF). Northeastern University. Retrieved 19 August 2014.
7. Hastie, T.; Tibshirani, R.; Friedman, J. H. (2009). "10. Boosting and Additive Trees". The Elements of Statistical Learning (2nd ed.). New York: Springer. pp. 337–384. ISBN 0-387-84857-6.
8. Note: in case of usual CART trees, the trees are fitted using least-squares loss, and so the coefficient b_{jm}

for the region R_{jm} is equal to just the value of output variable, averaged over all training instances in R_{jm}.

9. Note that this is different from bagging, which samples with replacement because it uses samples of the same size as the training set.
10. Ridgeway, Greg (2007). Generalized Boosted Models: A guide to the gbm package. (http://cran.r-project.org/web/packages/gbm/gbm.pdf)
11. Tianqi Chen. Introduction to Boosted Trees (http://homes.cs.washington.edu/~tqchen/pdf/BoostedTree.pdf)
12. Cossock, David and Zhang, Tong (2008). Statistical Analysis of Bayes Optimal Subset Ranking (http://www.stat.rutgers.edu/~tzhang/papers/it08-ranking.pdf), page 14.
13. Yandex corporate blog entry about new ranking model "Snezhinsk" (http://webmaster.ya.ru/replies.xml?item_no=5707&ncrnd=5118) (in Russian)

Retrieved from "https://en.wikipedia.org/w/index.php?title=Gradient_boosting&oldid=692320907"

Categories: Decision trees | Ensemble learning