Introduction to Machine Learning

Boosting - Advanced

Department of Statistics – LMU Munich

Gradient boosting and trees

GRADIENT BOOSTING AND TREES

Trees are mainly used as base learners for gradient boosting in ML. A great deal of research has been done on this combination so far, and it often provides the best results.

Reminder: Advantages of trees

- No problems with categorical features.
- No problems with outliers in feature values.
- No problems with missing values.
- No problems with monotone transformations of features.
- Trees (and stumps!) can be fitted quickly, even for large *n*.
- Trees have a simple built-in type of variable selection.

Gradient boosted trees retains all of them, and strongly improves the trees' predictive power. Furthermore, it is possible to adapt gradient boosting especially to tree learners.

GRADIENT BOOSTING AND TREES

One can write a tree as: $b(\mathbf{x}) = \sum_{t=1}^{T} c_t \mathbb{I}(\mathbf{x} \in R_t)$, where R_t are the terminal regions and c_t the corresponding means.

This special additive structure can be exploited by boosting:

$$\begin{aligned} f^{[m]}(\mathbf{x}) &= f^{[m-1]}(\mathbf{x}) + \beta^{[m]} b^{[m]}(\mathbf{x}) \\ &= f^{[m-1]}(\mathbf{x}) + \beta^{[m]} \sum_{t=1}^{T^{[m]}} c_t^{[m]} \mathbb{I}(\mathbf{x} \in R_t^{[m]}) \end{aligned}$$

Actually, we do not have to find $c_t^{[m]}$ and $\beta^{[m]}$ in two separate steps (fitting against pseudo-residuals, then line search). Also note that the $c_t^{[m]}$ will not really be loss-optimal as we used squared error loss to fit them against the pseudo residuals.

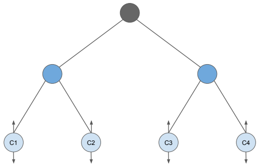
GRADIENT BOOSTING AND TREES

What we will do instead, is:

$$f^{[m]}(\mathbf{x}) = f^{[m-1]}(\mathbf{x}) + \sum_{j=1}^{T^{[m]}} \tilde{c}_t^{[m]} \mathbb{I}(\mathbf{x} \in R_t^{[m]})$$

We now induce the structure of the tree with respect to squared error loss, but then determine / change all $\tilde{c}_t^{[m]}$ individually and directly L-optimally:

$$\tilde{c}_t^{[m]} = \arg\min_{c} \sum_{\mathbf{x}^{(i)} \in R_t^{[m]}} L(y^{(i)}, f^{[m-1]}(\mathbf{x}^{(i)}) + c)$$



GRADIENT BOOSTING AND TREES

Algorithm 1 Gradient Tree Boosting Algorithm.

```
 1: Initialize $\hat{f}^{[0]}(\mathbf{x}) = \arg\min_\theta \sum_{i=1}^n L(y^{(i)}, \theta)$
 2: for $m = 1 \to M$ do
 3:     For all $i$: $r^{[m](i)} = -\left[\frac{\partial L\left(y^{(i)}, f\left(\mathbf{x}^{(i)}\right)\right)}{\partial f\left(\mathbf{x}^{(i)}\right)}\right]_{f=\hat{f}^{[m-1]}}$
 4:     Fit regr. tree to the $r^{[m](i)}$ giving terminal regions $R_t^{[m]},\ t = 1, \dots, T^{[m]}$
 5:     for $t = 1 \to T^{[m]}$ do
 6:         $\hat{c}_t^{[m]} = \arg\min_c \sum_{\mathbf{x}^{(i)} \in R_t^{[m]}} L(y^{(i)}, f^{[m-1]}(\mathbf{x}^{(i)}) + c)$
 7:     end for
 8:     $\hat{b}^{[m]}(\mathbf{x}) = \sum_{t=1}^T \hat{c}_t^{[m]} \mathbb{I}(\mathbf{x} \in R_t)$,
 9:     Update $\hat{f}^{[m]}(\mathbf{x}) = \hat{f}^{[m-1]}(\mathbf{x}) + \hat{b}^{[m]}(\mathbf{x})$
10: end for
11: Output $\hat{f}(\mathbf{x}) = \hat{f}^{[M]}(\mathbf{x})$
```

BINARY CLASSIFICATION

For $\mathcal{Y} = \{0, 1\}$, we simply have to select an appropriate loss function, so let's use binomial loss as in logistic regression:

$$L(y, f(\mathbf{x})) = -yf(\mathbf{x}) + ln(1 + \exp(f(\mathbf{x})))$$

Then,

$$\begin{aligned}\tilde{r} &= -\frac{\partial L(y, f(\mathbf{x}))}{\partial f(\mathbf{x})} \\ &= y - \frac{\exp(f(\mathbf{x}))}{1 + \exp(f(\mathbf{x}))} \\ &= y - \frac{1}{1 + \exp(-f(\mathbf{x}))} = y - s(f(\mathbf{x})).\end{aligned}$$

Here, $s(f(\mathbf{x}))$ is the logistic sigmoid function, applied to a scoring model. So effectively the pseudo residuals are $y - \pi(\mathbf{x})$.
Through $\pi(\mathbf{x}) = s(f(\mathbf{x}))$ we can also estimate posterior probabilities.

MULTI-CLASS PROBLEMS

We proceed as in softmax regression and model a categorical distribution, with multinomial loss. For $\mathcal{Y} = \{1, \dots, g\}$, we create g discriminant functions $f_k(x)$, one for each class, each an additive model of trees.
We define the $\pi_k(\mathbf{x})$ through the softmax function:

$$\pi_k(\mathbf{x}) = s_k(f_1(\mathbf{x}), \dots, f_g(\mathbf{x})) = \exp(f_k(x)) / \sum_{j=1}^{g} \exp(f_j(\mathbf{x}))$$

Multinomial loss L:

$$L(y, f_1(\mathbf{x}), \dots f_g(\mathbf{x})) = -\sum_{k=1}^{g} \mathbb{I}(y = k) \ln \pi_k(\mathbf{x})$$

And for the derivative the following holds:

$$-\frac{\partial L(y_k, f_1(\mathbf{x}), \dots, f_g(\mathbf{x}))}{\partial f_k(x)} = \mathbb{I}(y = k) - \pi_k(\mathbf{x})$$

MULTI-CLASS PROBLEMS

Determining the tree structure by squared-error-loss works just like before in the 2 class problem.

In the estimation of the c though, all the models depend on each other because of the definition of L. Optimizing this is more difficult, so we will skip the details and just present the results.

The estimated class for x is of course exactly the k for which $\pi_k(\mathbf{x})$ is maximal.

MULTI-CLASS PROBLEMS

Algorithm 2 Gradient Boosting for K-class Classification.

```
1: Initialize $f_k^{[0]}(\mathbf{x}) = 0,\ k = 1, \dots, g$
2: for $m = 1 \to M$ do
3:     Set $\pi_k(\mathbf{x}) = \frac{\exp(f_k(x))}{\sum_j \exp(f_j(x))}, k = 1, \dots, g$
4:     for $k = 1 \to g$ do
5:         For all $i$: Compute $r_k^{[m](i)} = \mathbb{I}(y^{(i)} = k) - \pi_k(\mathbf{x}^{(i)})$
6:         Fit regr. tree to the $r_k^{[m](i)}$ giving terminal regions $R_{tk}^{[m]}$
7:         Compute
8:             $\hat{c}_{tk}^{[m]} = \frac{g-1}{g} \frac{\sum_{\mathbf{x}^{(i)} \in R_{tk}^{[m]}} r_k^{[m](i)}}{\sum_{\mathbf{x}^{(i)} \in R_{tk}^{[m]}} \left|r_k^{[m](i)}\right| \left(1 - \left|r_k^{[m](i)}\right|\right)}$
9:         Update $\hat{f}_k^{[m]}(\mathbf{x}) = \hat{f}_k^{[m-1]}(x) + \sum_t \hat{c}_{tk}^{[m]} \mathbb{I}\left(x \in R_{tk}^{[m]}\right)$
10:    end for
11: end for
12: Output $\hat{f}_1^{[M]}, \dots, \hat{f}_g^{[M]}$
```

MULTI-CLASS PROBLEMS

Derivation of the algorithm:

- from Friedman, J. H. - Greedy Function Approximation: A Gradient Boosting Machine (1999)
- In each iteration m we calculate the pseudo residuals

$$r_k^{[m](i)} = \mathbb{I}(y^{(i)} = k) - \pi_k^{[m-1]}(\mathbf{x}^{(i)}),$$

  where $\pi_k^{[m-1]}(\mathbf{x}^{(i)})$ is derived from $f^{[m-1]}(\mathbf{x})$
- Thus, g trees are induced at each iteration m to predict the corresponding current pseudo residuals for each class on the probability scale.
- Each of these trees has T terminal nodes with corresponding regions $R_{tk}^{[m]}$.

MULTI-CLASS PROBLEMS

- The model updates $\hat{c}_{tk}^{[m]}$ corresponding to these regions are the solution to

$$\hat{c}_{tk}^{[m]} = \arg\min_{c} \sum_{i=1}^{n} \sum_{k=1}^{g} L\left(y_k^{(i)}, f^{[m-1]}(\mathbf{x}^{(i)}) + \sum_{t=1}^{T} \hat{c}_{tk} \mathbb{I}\left(\mathbf{x}^{(i)} \in R_t^{[m]} \right) \right)$$

  where L is the multinomial loss function $L(y, f_1(\mathbf{x}), \ldots f_g(\mathbf{x})) = -\sum_{k=1}^{g} \mathbb{I}(y = k) \ln \pi_k(\mathbf{x})$ and $\pi_k(\mathbf{x}) = \frac{\exp(f_k(\mathbf{x}))}{\sum_j \exp(f_j(\mathbf{x}))}$ as before.
- This has no closed form solution and additionally, the regions corresponding to the different class tress overlap, so that the solution does not reduce to a separate calculation within each region of each tree.

MULTI-CLASS PROBLEMS

- Hence, we approximate the solution with a single Newton-Raphson step, using a diagonal approximation to the Hessian.
- This decomposes the problem into a separate calculation for each terminal node of each tree.
- The result is

$$\hat{c}_{tk}^{[m]} = \frac{g-1}{g} \frac{\sum_{\mathbf{x}^{(i)} \in R_{tk}^{[m]}} r_k^{[m](i)}}{\sum_{\mathbf{x}^{(i)} \in R_{tk}^{[m]}} \left| r_k^{[m](i)} \right| \left(1 - \left| r_k^{[m](i)} \right| \right)}$$

- The update is then done by

$$\hat{f}_k^{[m]}(x) = \hat{f}_k^{[m-1]}(x) + \sum_t \hat{c}_{tk}^{[m]} \mathbb{I}\left(x \in R_{tk}^{[m]}\right)$$

REGULARIZATION AND SHRINKAGE

If GB runs for a long number of iterations, it can overfit due to its aggressive loss minimization.

Options for regularization

- Limit the number of boosting iterations M ("early stopping"), i.e., limit the number of additive components.
- Limit the depth of the trees. This can also be interpreted as choosing the order of interaction.
- Shorten the step length $\beta^{[m]}$ in each iteration.

The latter is achieved by multiplying $\beta^{[m]}$ with a small $\nu \in (0, 1]$:

$$f^{[m]}(\mathbf{x}) = f^{[m-1]}(\mathbf{x}) + \nu\beta^{[m]} b(\mathbf{x}, \theta^{[m]})$$

ν is called **shrinkage parameter** or **learning rate**.

REGULARIZATION AND SHRINKAGE

Obviously, the optimal values for M and ν strongly depend on each other: By increasing M one can use a smaller value for ν and vice versa.

In practice it is often recommended to choose ν quite small and choose M by cross-validation.

It's probably best to tune all three parameters jointly based on the training data via cross-validation or a related method.

STOCHASTIC GRADIENT BOOSTING

This is a minor modification to boosting to incorporate the advantages of bagging into the method. The idea was formulated quite early by Breiman.

Instead of fitting on all the data points, a random subsample is drawn in each iteration.

Especially for small training sets, this simple modification often leads to significant empirical improvements. How large the improvements are depends on data structure, size of the data set, base learner and size of the subsamples (so this is another tuning parameter).

EXAMPLE: SPAM DETECTION

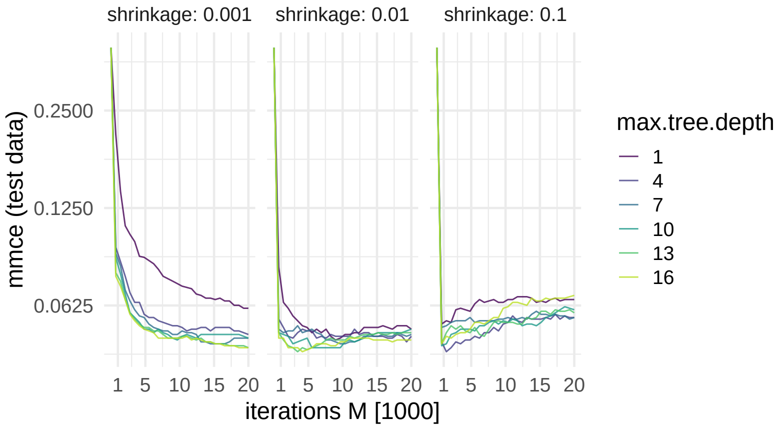
We fit a gradient boosting model for different parameter values

Parameter name	Values
distribution	Bernoulli (for classification)
shrinkage ν	$0.001, 0.01, 0.1$
number of trees *M*	$[0, \dots, 20000]$
max. tree depth	$1, 4, 7, 10, 13, 16$

We observe the error on a separate test set to find the optimal parameters.

EXAMPLE: SPAM DETECTION

Misclassification rate for different hyperparameter settings (shrinkage and maximum tree depth) of gradient boosting:



ADDITIONAL INFORMATION

By choosing a suitable loss function it is also possible to model a large number of different problem domains

- Regression
- (Multi-class) Classification
- Count data
- Survival data
- Ordinal data
- Quantile regression
- Ranking problems
- ...

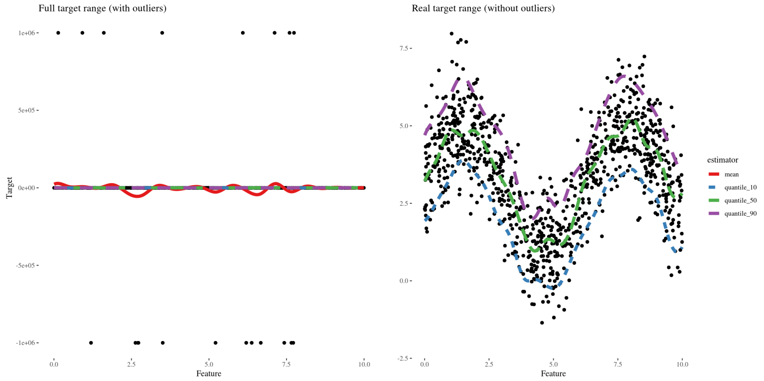
Different base learners increase flexibility (see componentwise gradient boosting). If we model only individual variables, the resulting regularized variable selection is closely related to L1 regularization.

ADDITIONAL INFORMATION

For example, using the pinball loss in boosting

$$L(y, f(\mathbf{x})) = \begin{cases} (1-\alpha)(f(\mathbf{x}) - y), & \text{if } y < f(\mathbf{x}) \\ \alpha(y - f(\mathbf{x})), & \text{if } y \geq f(\mathbf{x}) \end{cases}$$

models the α-quantiles:



ADDITIONAL INFORMATION

The AdaBoost fit has the structure of an additive model with “basis functions” $b^{[m]}(x)$.

It can be shown (see Hastie et al. 2009, chapter 10) that AdaBoost corresponds to minimizing the empirical risk in each iteration m using the *exponential* loss function:

$$\begin{aligned} L(y, \hat{f}^{[m]}(\mathbf{x})) &= \exp\left(-y\hat{f}^{[m]}(\mathbf{x})\right) \\ \mathcal{R}_{\text{emp}}(\hat{f}^{[m]}) &= \sum_{i=1}^{n} L(y^{(i)}, \hat{f}^{[m]}(\mathbf{x}^{(i)})) \\ &= \sum_{i=1}^{n} L(y^{(i)}, \hat{f}^{[m-1]}(\mathbf{x}^{(i)}) + \beta b(\mathbf{x}^{(i)})) \end{aligned}$$

with minimizing over β and b and where $\hat{f}^{[m-1]}$ is the boosting fit in iteration $m-1$.

TAKE HOME MESSAGE

Gradient boosting is a statistical reinterpretation of the older AdaBoost algorithm.

Base learners are added in a “greedy” fashion, so that they point in the direction of the negative gradient of the empirical risk.

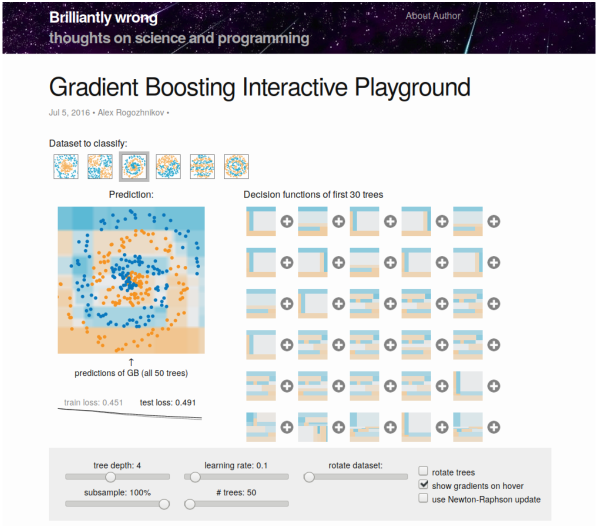
Regression base learners are fitted even for classification problems.

Often the base learners are (shallow) trees, but arbitrary base learners are possible.

The method can be adjusted flexibly by changing the loss function, as long as it’s differentiable.

Methods to evaluate variable importance and to do variable selection exist.

GRADIENT BOOSTING PLAYGROUND



Brilliantly wrong
thoughts on science and programming

About Author

Gradient Boosting Interactive Playground

Jul 5, 2016 • Alex Rogozhnikov •

Dataset to classify:

Prediction:

Decision functions of first 30 trees

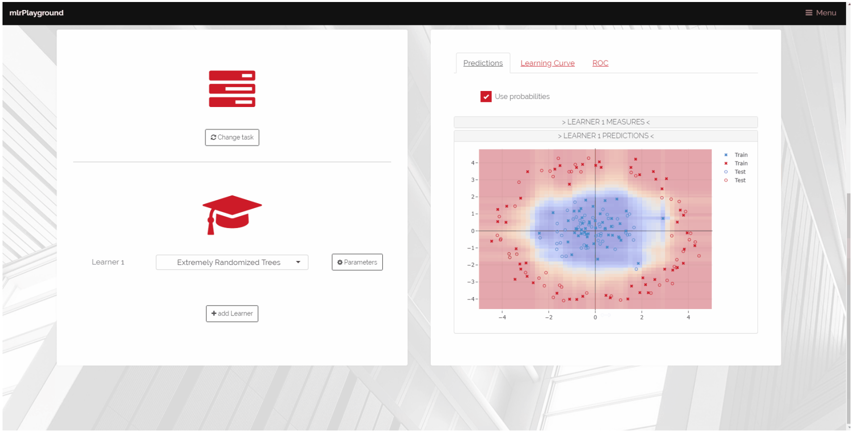
↑
predictions of GB (all 50 trees)

train loss: 0.451 test loss: 0.491

tree depth: 4
learning rate: 0.1
rotate dataset:
subsample: 100%
# trees: 50

☐ rotate trees
☑ show gradients on hover
☐ use Newton-Raphson update

MLRPLAYGROUND



Open in browser.