Lecture 10: Boosting

STATS 202: Statistical Learning and Data Science

Linh Tran
tranlm@stanford.edu

Department of Statistics
Stanford University

July 28, 2025

Announcements

- Midterm grades will be out soon
- Homework 3 due today
  - Homework 4 is posted
- Final predictions due in 2 weeks
- Section this Friday goes into more final project details

Outline

- Boosting introduction
- Boosting vs bagging
- Boosting remarks
- AdaBoost
- Boosting training error
- Gradient boosting
- Regularization
- Random tips

Recall

- Decision trees partition our feature space and make predictions within each partitioned region.
- Bagging reduces the high variability of decision trees.
- Random forest further reduces variance via random variable selection.

Recall

- Decision trees partition our feature space and make predictions within each partitioned region.
- Bagging reduces the high variability of decision trees.
- Random forest further reduces variance via random variable selection.

Question: Is there another way of improving the performance of decision trees?

Boosting

Boosting uses a set of weak learners (e.g. decision trees) to create a strong one.

Boosting

Boosting uses a set of weak learners (e.g. decision trees) to create a strong one.

The general algorithm is:

1. Fit an initial $\hat{f}_n^0$ to the data and compute residuals r_i.

Boosting

Boosting uses a set of weak learners (e.g. decision trees) to create a strong one.

The general algorithm is:

1. Fit an initial $\hat{f}_n^0$ to the data and compute residuals r_i.
2. For $b = 1, ..., B$:
   - Fit a weak leaner $\hat{f}_n^b$ on the residuals.

Boosting

Boosting uses a set of weak learners (e.g. decision trees) to create a strong one.

The general algorithm is:

1. Fit an initial $\hat{f}_n^0$ to the data and compute residuals r_i.
2. For $b = 1, ..., B$:
   - Fit a weak leaner $\hat{f}_n^b$ on the residuals.
   - With learning rate λ_b, update prediction to:

$$\hat{f}_n \leftarrow \hat{f}_n + \lambda_b \hat{f}_n^b. \tag{1}$$

Boosting

Boosting uses a set of weak learners (e.g. decision trees) to create a strong one.

The general algorithm is:

1. Fit an initial $\hat{f}_n^0$ to the data and compute residuals r_i.
2. For $b = 1, ..., B$:
   - Fit a weak leaner $\hat{f}_n^b$ on the residuals.
   - With learning rate λ_b, update prediction to:
     $$\hat{f}_n \leftarrow \hat{f}_n + \lambda_b \hat{f}_n^b. \tag{1}$$
   - Update the residuals
     $$r_i \leftarrow r_i - \lambda_b \hat{f}_n^b(x_i). \tag{2}$$

Boosting

Boosting uses a set of weak learners (e.g. decision trees) to create a strong one.

The general algorithm is:

1. Fit an initial $\hat{f}_n^0$ to the data and compute residuals r_i.
2. For $b = 1, ..., B$:
   - Fit a weak leaner $\hat{f}_n^b$ on the residuals.
   - With learning rate λ_b, update prediction to:
     $$\hat{f}_n \leftarrow \hat{f}_n + \lambda_b \hat{f}_n^b. \tag{1}$$
   - Update the residuals
     $$r_i \leftarrow r_i - \lambda_b \hat{f}_n^b(x_i). \tag{2}$$
3. Output prediction, e.g.
   $$\hat{f}_n(x) = \sum_{b=1}^{B} \lambda_b \hat{f}_n^b(x). \tag{3}$$

Boosting

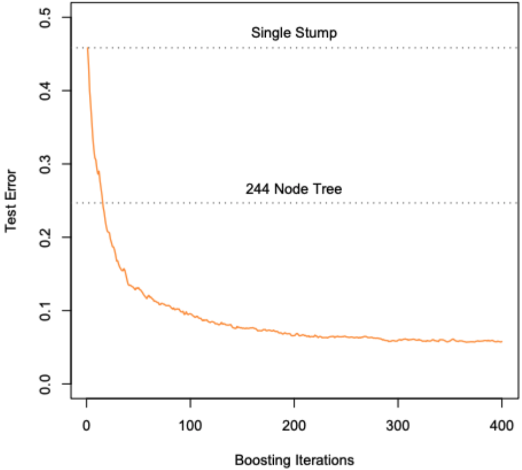


Figure 10.2

Boosting hyperparameters

Hyper-parameters to consider when applying a boosting model:

- The number of learners (aka trees) B to use.
- The shrinkage parameter λ_b.
- The parameters of the learner (e.g. splits in each tree).

Typically, these are found via *cross-validation*.

Boosting vs bagging

Bagging: For $b = 1, \ldots, B$:

1. Created a bootstrapped sample, P_n^b.
2. Get estimate $\hat{f}_n^b(x)$ using P_n^b.

Average the estimates, i.e.

$$\hat{f}_n^{\text{bag}}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}_n^b(x).$$

- P_n is varied for each fit.
- Designed to reduce variance.

Boosting vs bagging

Bagging: For $b = 1, \ldots, B$:

1. Created a bootstrapped sample, P_n^b.
2. Get estimate $\hat{f}_n^b(x)$ using P_n^b.

Average the estimates, i.e.

$$\hat{f}_n^{\text{bag}}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}_n^b(x).$$

- P_n is varied for each fit.
- Designed to reduce variance.

Boosting: For $b = 1, \ldots, B$:

1. Get estimate $\hat{f}_n^b(x)$ for the residuals r^{b-1}.
2. Update residuals $r_i^b = r_i^{b-1} - \lambda_b \hat{f}_n^b(x_i)$.

Sum the estimates, i.e.

$$\hat{f}_n^{\text{boost}}(x) = \sum_{b=1}^{B} \lambda_b \hat{f}_n^b(x).$$

- 'Y' is varied for each fit.
- Designed to reduce bias.

Remarks:

- Boosting has been called the "best off-the-shelf classifier in the world".

Remarks:

- Boosting has been called the "best off-the-shelf classifier in the world".
- Boosting (generally) works by upweighing points at each iteration which are misclassified.

Remarks:

- Boosting has been called the "best off-the-shelf classifier in the world".
- Boosting (generally) works by upweighing points at each iteration which are misclassified.
- Boosting can use any classifier as its weak learner (base classifier) but decision trees are by far the most popular.

Boosting

Remarks:

- Boosting has been called the "best off-the-shelf classifier in the world".
- Boosting (generally) works by upweighing points at each iteration which are misclassified.
- Boosting can use any classifier as its weak learner (base classifier) but decision trees are by far the most popular.
- Boosting learns slowly, first using the samples that are easiest to predict, then slowly down weigh these cases, moving on to harder samples.

Remarks:

- Boosting has been called the "best off-the-shelf classifier in the world".
- Boosting (generally) works by upweighing points at each iteration which are misclassified.
- Boosting can use any classifier as its weak learner (base classifier) but decision trees are by far the most popular.
- Boosting learns slowly, first using the samples that are easiest to predict, then slowly down weigh these cases, moving on to harder samples.
- Boosting can give zero training error, but rarely overfits.

Remarks:

- Boosting has been called the "best off-the-shelf classifier in the world".
- Boosting (generally) works by upweighing points at each iteration which are misclassified.
- Boosting can use any classifier as its weak learner (base classifier) but decision trees are by far the most popular.
- Boosting learns slowly, first using the samples that are easiest to predict, then slowly down weigh these cases, moving on to harder samples.
- Boosting can give zero training error, but rarely overfits.
- Can be thought of as fitting a model on multiple data sets.

AdaBoost

A popular earlier version of boosting is *AdaBoost*:

1. Initialize the observation weights $w_i = 1/n : i = 1, ..., n$.

AdaBoost

A popular earlier version of boosting is *AdaBoost*:

1. Initialize the observation weights $w_i = 1/n : i = 1, ..., n$.
2. For $b = 1, ..., B$:
   a Fit a classifier $G^b(x)$ to the training data using weights w_i.

AdaBoost

A popular earlier version of boosting is *AdaBoost*:

1. Initialize the observation weights $w_i = 1/n : i = 1, ..., n$.
2. For $b = 1, ..., B$:
   a. Fit a classifier $G^b(x)$ to the training data using weights w_i.
   b. Compute

$$err_b = \frac{\sum_{i=1}^{n} w_i \mathbb{I}(y_i \neq G^b(x_i))}{\sum_{i=1}^{n} w_i} \tag{4}$$

A popular earlier version of boosting is *AdaBoost*:

1. Initialize the observation weights $w_i = 1/n : i = 1, ..., n$.
2. For $b = 1, ..., B$:
   a. Fit a classifier $G^b(x)$ to the training data using weights w_i.
   b. Compute
   $$err_b = \frac{\sum_{i=1}^n w_i \mathbb{I}(y_i \neq G^b(x_i))}{\sum_{i=1}^n w_i} \tag{4}$$
   c. Compute $\alpha_b = \log((1 - err_b)/err_b)$

A popular earlier version of boosting is *AdaBoost*:

1. Initialize the observation weights $w_i = 1/n : i = 1, ..., n$.
2. For $b = 1, ..., B$:
   a. Fit a classifier $G^b(x)$ to the training data using weights w_i.
   b. Compute
   $$err_b = \frac{\sum_{i=1}^n w_i \mathbb{I}(y_i \neq G^b(x_i))}{\sum_{i=1}^n w_i} \tag{4}$$
   c. Compute $\alpha_b = \log((1 - err_b)/err_b)$
   d. Set $w_i \leftarrow w_i \cdot \exp[\alpha_b \mathbb{I}(y_i \neq G^b(x_i))] : i = 1, ..., n$.

AdaBoost

A popular earlier version of boosting is *AdaBoost*:

1. Initialize the observation weights $w_i = 1/n : i = 1, ..., n$.
2. For $b = 1, ..., B$:
   - a Fit a classifier $G^b(x)$ to the training data using weights w_i.
   - b Compute

$$err_b = \frac{\sum_{i=1}^{n} w_i \mathbb{I}(y_i \neq G^b(x_i))}{\sum_{i=1}^{n} w_i} \tag{4}$$

   - c Compute $\alpha_b = \log((1 - err_b)/err_b)$
   - d Set $w_i \leftarrow w_i \cdot \exp[\alpha_b \mathbb{I}(y_i \neq G^b(x_i))] : i = 1, ..., n$.
3. Output $G_B(x) = \text{sign}\left(\sum_{b=1}^{B} \alpha_b G^b(x)\right)$.

Boosting

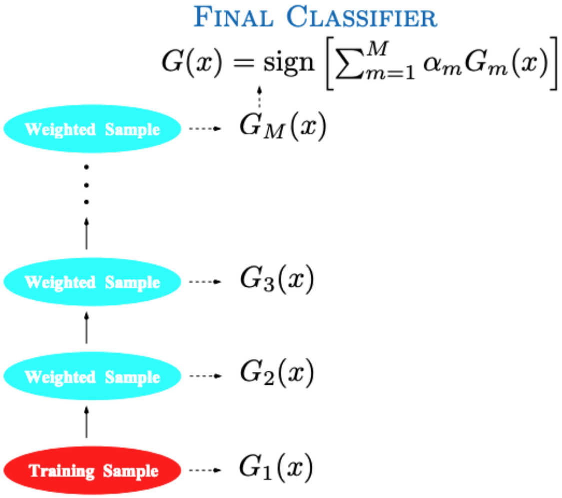


Figure 10.1

AdaBoost example

Sonar Data (n=208)

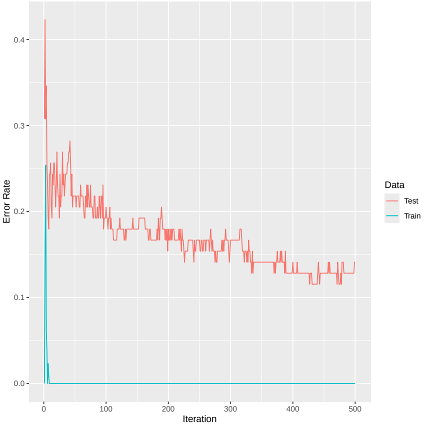
Features

- Measured energy level within one of 60 frequency bands

Label

- Indicator object is a Mine (1) vs Rock (0)

AdaBoost example



AdaBoost applied to the *Sonar Data*.

Training error

Question: What happens after the training error reaches 0?

Training error

Question: What happens after the training error reaches 0?

Define:

$$G_B^*(x) = \frac{\sum_{b=1}^{B} \alpha_b G^b(x)}{\sum_{b=1}^{B} \alpha_b} \tag{5}$$

Training error

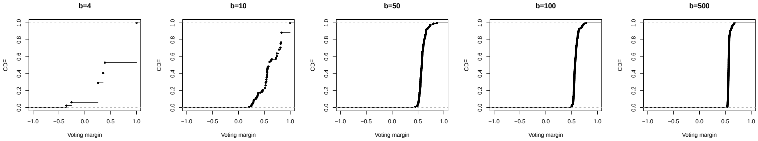
Question: What happens after the training error reaches 0?

Define:

$$G_B^*(x) = \frac{\sum_{b=1}^{B} \alpha_b G^b(x)}{\sum_{b=1}^{B} \alpha_b} \tag{5}$$

We can look at *voting margins* for our training data, i.e.

$$margin(x) = y * G_B^*(x) \tag{6}$$



Training error

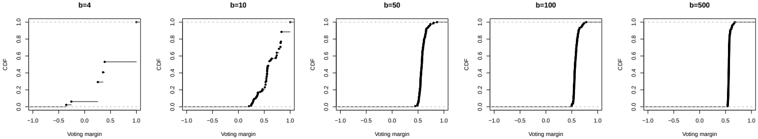
Question: What happens after the training error reaches 0?

Define:

$$G_B^*(x) = \frac{\sum_{b=1}^{B} \alpha_b G^b(x)}{\sum_{b=1}^{B} \alpha_b} \quad (5)$$

We can look at *voting margins* for our training data, i.e.

$$margin(x) = y * G_B^*(x) \quad (6)$$



n.b. Letting $err_b \leq 1/2 - \gamma$, then $\text{Error}_{train} \leq (\sqrt{1 - 4\gamma^2})^B$

Weak learners

Question: How “weak” should our learners be?

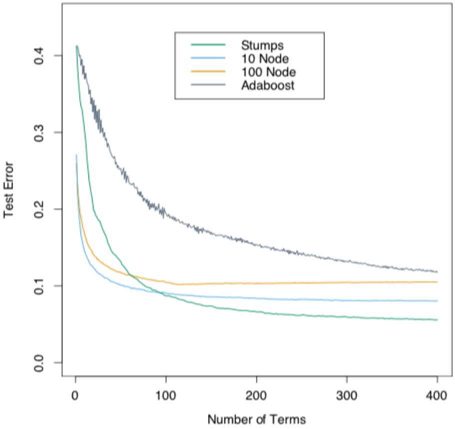
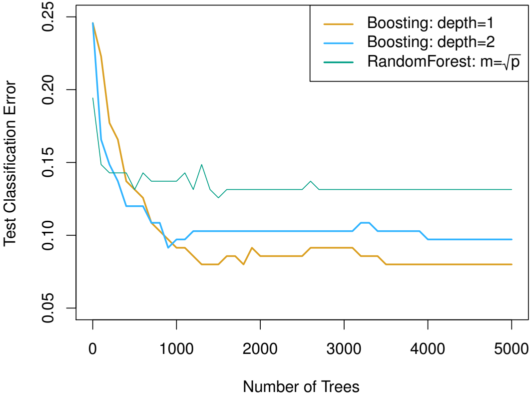


Figure 10.9

Example: Applied to 15-class gene expression data (Figure 8.11).



Forward Stagewise Additive Modeling

Boosting can be viewed as an additive model, i.e.

$$f(\mathbf{X}) = \sum_{m=1}^{M} \beta_m b(\mathbf{X}; \gamma_m) \tag{7}$$

for M basis functions characterized by γ_m

Forward Stagewise Additive Modeling

Boosting can be viewed as an additive model, i.e.

$$f(\mathbf{X}) = \sum_{m=1}^{M} \beta_m b(\mathbf{X}; \gamma_m) \tag{7}$$

for M basis functions characterized by γ_m

Algorithm 10.2 *Forward Stagewise Additive Modeling.*

1. Initialize $f_0(x) = 0$.
2. For $m = 1$ to M:
   (a) Compute
   $$(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)).$$
   (b) Set $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$.

Forward Stagewise Additive Modeling

Algorithm 10.2 *Forward Stagewise Additive Modeling.*

1. Initialize $f_0(x) = 0$.
2. For $m = 1$ to M:
   (a) Compute
   $$(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)).$$
   (b) Set $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$.

$$\begin{aligned} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)) &= (y_i - (f_{m-1}(x_i) + \beta b(x_i; \gamma)))^2 \\ &= (y_i - f_{m-1}(x_i) - \beta b(x_i; \gamma))^2 \\ &= (r_{im} - \beta b(x_i; \gamma))^2 \end{aligned}$$

Forward Stagewise Additive Modeling

Algorithm 10.2 *Forward Stagewise Additive Modeling.*

1. Initialize $f_0(x) = 0$.
2. For $m = 1$ to M:
   (a) Compute
   $$(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)).$$
   (b) Set $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$.

$$\begin{aligned} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)) &= (y_i - (f_{m-1}(x_i) + \beta b(x_i; \gamma)))^2 \\ &= (y_i - f_{m-1}(x_i) - \beta b(x_i; \gamma))^2 \\ &= (r_{im} - \beta b(x_i; \gamma))^2 \end{aligned}$$

Problem: RSS isn't a good loss for classification

Define the following loss:

$$L(y, f(x)) = \exp(-yf(x)) \tag{8}$$

Applying the forward stagewise additive representation to Adaboost:

$$\begin{aligned}(\beta_m, G_m) &= \arg\min_{\beta, G} \sum_{i=1}^{n} \exp\left(-y_i(f_{m-1}(x_i) + \beta G(x_i))\right) \\ &= \arg\min_{\beta, G} \sum_{i=1}^{n} w_i^{(m)} \exp\left(-\beta y_i G(x_i)\right)\end{aligned}$$

where $w_i^{(m)} = \exp(-y_i f_{m-1}(x_i))$

It can be shown that

$$w_i^{(m+1)} = w_i^{(m)} \cdot \exp(\alpha_m \mathbb{I}(y_i \neq G_m(x_i))) \cdot \exp(-\beta_m)$$

Gradient Boosting

Gradient boosting generalizes $L(y, f(x))$ to any smooth loss function.

Some common loss functions:

TABLE 10.2. *Gradients for commonly used loss functions.*

Setting	Loss Function	$-\partial L(y_i, f(x_i))/\partial f(x_i)$
Regression	$\frac{1}{2}[y_i - f(x_i)]^2$	$y_i - f(x_i)$
Regression	$\lvert y_i - f(x_i)\rvert$	$\text{sign}[y_i - f(x_i)]$
Regression	Huber	$y_i - f(x_i)$ for $\lvert y_i - f(x_i)\rvert \leq \delta_m$ $\delta_m \text{sign}[y_i - f(x_i)]$ for $\lvert y_i - f(x_i)\rvert > \delta_m$ where $\delta_m = \alpha\text{th-quantile}\{\lvert y_i - f(x_i)\rvert\}$
Classification	Deviance	kth component: $I(y_i = \mathcal{G}_k) - p_k(x_i)$

Gradient Boosting algorithm

1. Initialize $f_n^0(x) = \arg\min_{\gamma} \sum_{i=1}^n L(y_i, \gamma)$.

Gradient Boosting algorithm

1. Initialize $f_n^0(x) = \arg\min_\gamma \sum_{i=1}^n L(y_i, \gamma)$.
2. For $b = 1, ..., B$:

   a Compute the residuals/gradients:

$$r_i^b = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f^{b-1}} : i = 1, ..., n \tag{9}$$

Gradient Boosting algorithm

1. Initialize $f_n^0(x) = \arg\min_{\gamma} \sum_{i=1}^n L(y_i, \gamma)$.
2. For $b = 1, ..., B$:

   a Compute the residuals/gradients:

   $$r_i^b = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f^{b-1}} : i = 1, ..., n \tag{9}$$

   b Fit a regression tree to r_i^b, giving terminal regions $R_j^b : j = 1, ..., J^b$

Gradient Boosting algorithm

1. Initialize $f_n^0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma)$.
2. For $b = 1, ..., B$:
   a Compute the residuals/gradients:

$$r_i^b = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f^{b-1}} : i = 1, ..., n \tag{9}$$

   b Fit a regression tree to r_i^b, giving terminal regions $R_j^b : j = 1, ..., J^b$

   c For $j = 1, ..., J^b$, compute

$$\gamma_i^b = \arg\min_{\gamma} \sum_{x_i \in R_j^b} L(y_i, f^{b-1}(x_i) + \gamma) \tag{10}$$

Gradient Boosting algorithm

1. Initialize $f_n^0(x) = \underset{\gamma}{\arg\min} \sum_{i=1}^n L(y_i, \gamma)$.
2. For $b = 1, ..., B$:

   a Compute the residuals/gradients:

   $$r_i^b = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f^{b-1}} : i = 1, ..., n \tag{9}$$

   b Fit a regression tree to r_i^b, giving terminal regions $R_j^b : j = 1, ..., J^b$

   c For $j = 1, ..., J^b$, compute

   $$\gamma_i^b = \underset{\gamma}{\arg\min} \sum_{x_i \in R_j^b} L(y_i, f^{b-1}(x_i) + \gamma) \tag{10}$$

   d Update $f^b(x) = f^{b-1}(x) + \sum_{j=1}^{J^b} \gamma_j^b \mathbb{I}(x \in R_j^b)$

Gradient Boosting algorithm

1. Initialize $f_n^0(x) = \arg\min_{\gamma} \sum_{i=1}^n L(y_i, \gamma)$.
2. For $b = 1, ..., B$:

   a Compute the residuals/gradients:

   $$r_i^b = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f^{b-1}} : i = 1, ..., n \tag{9}$$

   b Fit a regression tree to r_i^b, giving terminal regions $R_j^b : j = 1, ..., J^b$

   c For $j = 1, ..., J^b$, compute

   $$\gamma_i^b = \arg\min_{\gamma} \sum_{x_i \in R_j^b} L(y_i, f^{b-1}(x_i) + \gamma) \tag{10}$$

   d Update $f^b(x) = f^{b-1}(x) + \sum_{j=1}^{J^b} \gamma_j^b \mathbb{I}(x \in R_j^b)$
3. Output $\hat{f}_n(x) = f^B(x)$.

Regularization

Gradient boosting is greedy and can still overfit.

Regularization

Gradient boosting is greedy and can still overfit.

Regularization methods:

- Tree constraints: for each tree, limiting the e.g. number of trees, depth, terminal nodes, observations in a split, improvement made.

Regularization

Gradient boosting is greedy and can still overfit.

Regularization methods:

- Tree constraints: for each tree, limiting the e.g. number of trees, depth, terminal nodes, observations in a split, improvement made.
- Shrinkage: Each tree is weighted to slow down the learning by the algorithm.

Regularization

Gradient boosting is greedy and can still overfit.

Regularization methods:

- Tree constraints: for each tree, limiting the e.g. number of trees, depth, terminal nodes, observations in a split, improvement made.
- Shrinkage: Each tree is weighted to slow down the learning by the algorithm.
- Random splitting: at each iteration a subsample of the training data is drawn at random (without replacement).

Regularization

Gradient boosting is greedy and can still overfit.

Regularization methods:

- Tree constraints: for each tree, limiting the e.g. number of trees, depth, terminal nodes, observations in a split, improvement made.
- Shrinkage: Each tree is weighted to slow down the learning by the algorithm.
- Random splitting: at each iteration a subsample of the training data is drawn at random (without replacement).
- Penalized learning: Apply $L1$ or $L2$ regularization to the terminal nodes.

Example: Shrinkage (ν)

$$f^b(x) = f^{b-1}(x) + \nu \sum_{j=1}^{J^b} \gamma_j^b \mathbb{I}(x \in R_j^b) \tag{11}$$

Further trades off slower / longer learning.

Example: Comparing shrinkage vs not.

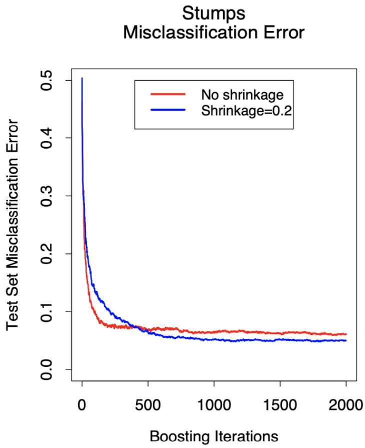


Figure 10.11

Sub-sampling

Example: Shrinkage with sub-sampling.

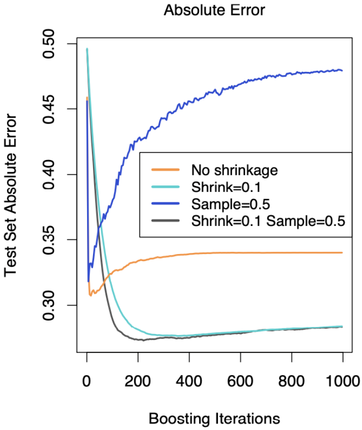


Figure 10.12

Gradient boosting tips

Gradient boosting wins most of the Kaggle competitions.

- Trick is to fine tune the hyper-parameters during training.

Gradient boosting tips

Gradient boosting wins most of the Kaggle competitions.

- Trick is to fine tune the hyper-parameters during training.

Some tips from Kaggle master *Owen Zhang*:

GBDT Hyper Parameter Tuning

Hyper Parameter	Tuning Approach	Range	Note
# of Trees	Fixed value	100-1000	Depending on datasize
Learning Rate	Fixed => Fine Tune	[2 - 10] / # of Trees	Depending on # trees
Row Sampling	Grid Search	[.5, .75, 1.0]	
Column Sampling	Grid Search	[.4, .6, .8, 1.0]	
Min Leaf Weight	Fixed => Fine Tune	3/(% of rare events)	Rule of thumb
Max Tree Depth	Grid Search	[4, 6, 8, 10]	
Min Split Gain	Fixed	0	Keep it 0

Best GBDT implementation today: https://github.com/tqchen/xgboost
by **Tianqi Chen** (U of Washington)

DataRobot

References

[1] ISL. Chapter 8

[2] ESL. Chapter 10

[3] Schapire, RE. The Boosting Approach to Machine Learning An Overview. Nonlinear Estimation and Classification, Springer, 2003.