Lecture 11: Boosting

STATS 202: Data Mining and Analysis

Linh Tran

tranlm@stanford.edu

Department of Statistics
Stanford University

August 2, 2021

Announcements

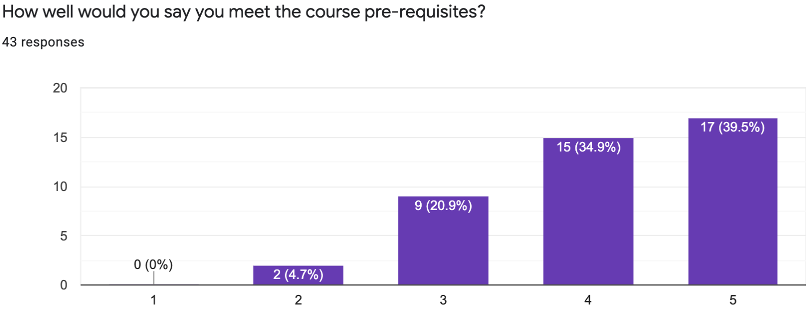
- Homework 3 is due this Friday.
- Final project review session this Friday.
  - Will be conducted like casual lab / office hours.
- Midterms are graded (regrades open for a week).

Outline

- Survey results
- Boosting

Survey (62% response)

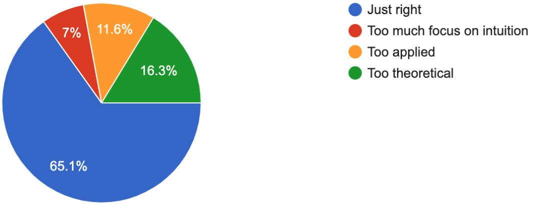
Course pre-requisites



Survey (62% response)
Course composition

How do you feel about the combination of course material overall (including lectures, homework, and exams)?

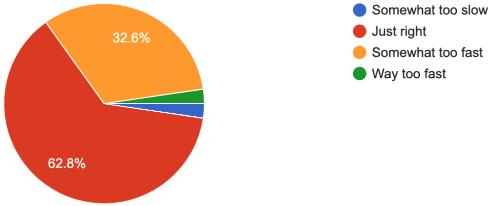
43 responses

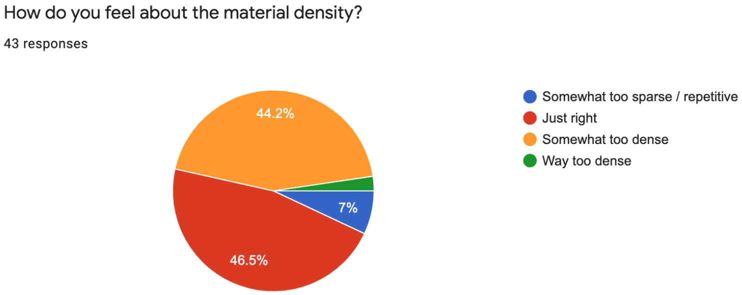


Survey (62% response)
Course pace

How do you feel about the course pace?

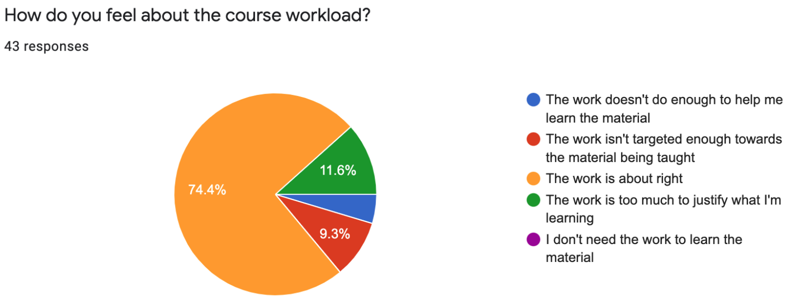
43 responses





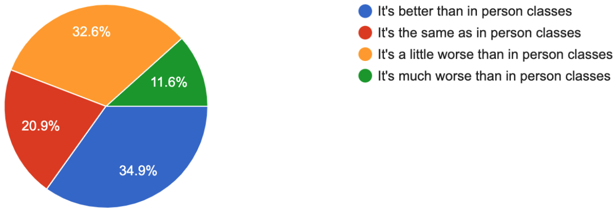
How do you feel about the material density?
43 responses
44.2%
7%
46.5%
Somewhat too sparse / repetitive
Just right
Somewhat too dense
Way too dense

Survey (62% response)
Course workload



How do you feel about completing the course remotely?

43 responses



- Can I have dedicated office hours?
- Office hours can be more in depth.
- Instructor / TA's should sync on HW questions.

Survey (62% response)
Remote feedback (paraphrased)

- Can we draw on a board?
- Add breaks for the Zoom lectures.
- Repeat questions before answering them.
- Try not to go over lecture time.
- Make announcements over Canvas as well.
- We're losing attention.

- Great course. Thank you TAs.
- Give more info on final project.
- The midterm was harder than the HW assignments.
- The lectures are/aren't built enough around textbook.
- Better expectation for HW answers, points are disproportionate, proofs derivations aren't fair.
- Stack exchange says that the team & textbook are wrong.

- Recent developments in the field, studies, etc.
- Real life problem, practical examples, COVID data, statistical errors in published research, industry applications.
- Review course material.
- Working with time series data/forecasting.
- Ideas and resources for continued learning in this area after this course.
- Derivations of the formulas presented in lectures.
- Embeddings? Ensemble Methods? NLP/text processing?
- Distributed computing / programming frameworks.

Survey (62% response)
Extra lectures

- **Review course material.**
- **Recent developments in the field, studies, etc.**
- **Real life problem, practical examples, COVID data, statistical errors in published research, industry applications.**
- Working with time series data/forecasting.
- Ideas and resources for continued learning in this area after this course.
- Derivations of the formulas presented in lectures.
- Embeddings? Ensemble Methods? NLP/text processing?
- Distributed computing / programming frameworks.

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.

$$\hat{f}_n(x) = \hat{f}_n^0 + \sum_{b=1}^{B} \lambda_b \hat{f}_n^b(x). \tag{3}$$

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 $b = 1, \dots, B$:

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".
- 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.

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

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

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 $\lambda_b = \log((1 - err_b)/err_b)$

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 $\lambda_b = \log((1 - err_b)/err_b)$
   - d Set $w_i \leftarrow w_i \cdot \exp[\lambda_b \mathbb{I}(y_i \neq G^b(x_i))] : i = 1, ..., n$.

AdaBoost

A popular earlier version of boosting is *AdaBoost*:

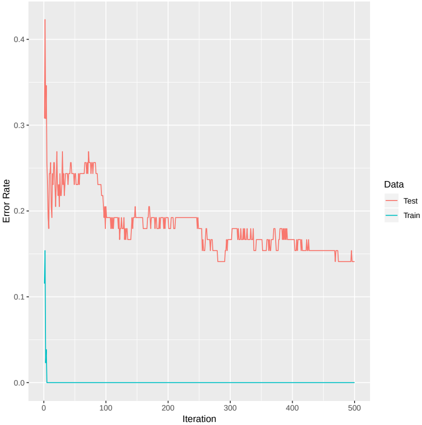
1. Initialize the observation weights $w_i = 1/n : i = 1, ..., n$.
2. For $b = 1, ..., B$:
   a Fit a classifier $G^b(x)$ to the training data using weights w_i.

   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 $\lambda_b = \log((1 - err_b)/err_b)$

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

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:

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

Training error

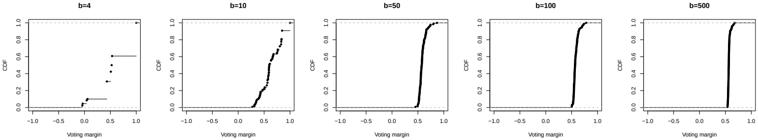
Question: What happens after the training error reaches 0?

Define:

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

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

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



Gradient Boosting

AdaBoost can be framed as 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 $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$.

where $L(y, f_{m-1}(x) + \beta b(x; \gamma))$ is the exponential loss, i.e.

$$L(y, f(x)) = \exp(-yf(x)) \tag{7}$$

Gradient Boosting

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

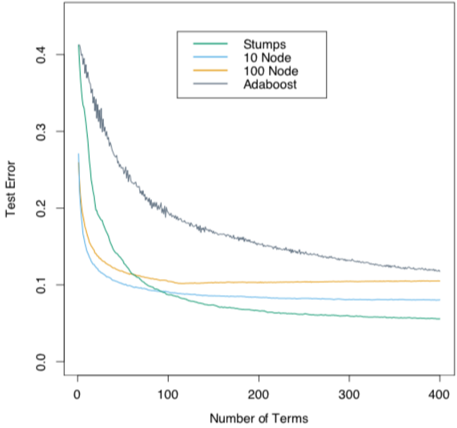
Some common loss functions:

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

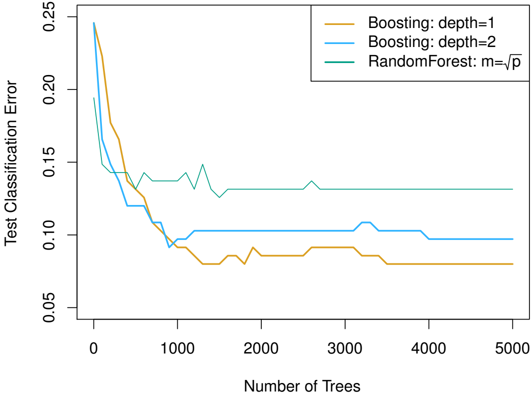
Setting	Loss Function	$-\partial L(y_i, f(x_i))/\partial f(x_i)$
Regression	$\frac{1}{2}[y_i - f(x_i)]^2$	$y_i - f(x_i)$
Regression	$\lvert y_i - f(x_i)\rvert$	$\text{sign}[y_i - f(x_i)]$
Regression	Huber	$y_i - f(x_i)$ for $\lvert y_i - f(x_i)\rvert \le \delta_m$ $\delta_m \text{sign}[y_i - f(x_i)]$ 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: $I(y_i = \mathcal{G}_k) - p_k(x_i)$

Boosting vs. AdaBoost

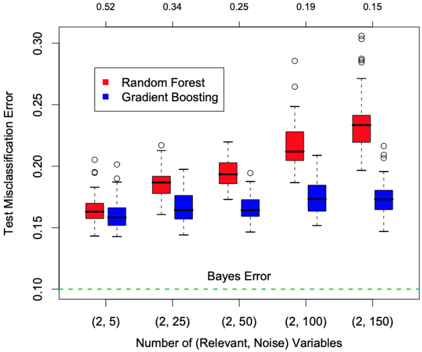
Example: Applied to simulated data.



Example: Applied to 15-class gene expression data.



Overfitting with Gradient boosting



Regularization

Gradient boosting is greedy and can quickly overfit.

Regularization

Gradient boosting is greedy and can quickly 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 quickly 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 quickly 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 quickly overfit.

Regularization methods:

- Tree constraints: for each tree, limiting the e.g. number of trees, depth, terminal nodes, obserations 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.

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.