On Gradient Boosting Machines

a.k.a. A Tale of Boosting

Hari A Ravindran

June 16, 2017

State of the art

- Enormous data sets are the norm nowadays.

State of the art

- Enormous data sets are the norm nowadays.
- Can scale software to fit a collection of linear and/or GLMs; however, predictive power suffers.

State of the art

- Enormous data sets are the norm nowadays.
- Can scale software to fit a collection of linear and/or GLMs; however, predictive power suffers.
- Need arose for general purpose tools that scale well to bigger problems.

State of the art

- Enormous data sets are the norm nowadays.
- Can scale software to fit a collection of linear and/or GLMs; however, predictive power suffers.
- Need arose for general purpose tools that scale well to bigger problems.
- Enter: Random forests and boosting. Both represent the fitted model by a sum of regression trees.

State of the art (contd.)

- **Random forest** Grow many deep regression trees to 'randomized' versions of the training data.

State of the art (contd.)

- **Random forest** Grow many deep regression trees to 'randomized' versions of the training data.
  - Basic mechanism: Variance reduction by averaging.

State of the art (contd.)

- **Random forest** Grow many deep regression trees to 'randomized' versions of the training data.
  - <u>Basic mechanism:</u> Variance reduction by averaging.
- **Boosting** Repeatedly grow shallow trees to the residuals. Then build up an additive model consisting of additive trees.

State of the art (contd.)

- **Random forest** Grow many deep regression trees to ‘randomized’ versions of the training data.
  - Basic mechanism: Variance reduction by averaging.
- **Boosting** Repeatedly grow shallow trees to the residuals. Then build up an additive model consisting of additive trees.
  - Basic mechanism: Bias reduction (though some flavours have to deal with variance reduction as well).

State of the art (contd.)

- **Random forest** Grow many deep regression trees to 'randomized' versions of the training data.
  - Basic mechanism: Variance reduction by averaging.
- **Boosting** Repeatedly grow shallow trees to the residuals. Then build up an additive model consisting of additive trees.
  - Basic mechanism: Bias reduction (though some flavours have to deal with variance reduction as well).
- Both methods inherit most of the good attributes of trees.

Adaboost

- The idea of boosting first conceived in 1980's.

Adaboost

- The idea of boosting first conceived in 1980's.
  - *Is weakly learnability equivalent to strong learnability?* by Kearns and Valiant (ACM Symposium on the Theory of Computing 1989)

Adaboost

- The idea of boosting first conceived in 1980's.
  - *Is weakly learnability equivalent to strong learnability?* by Kearns and Valiant (ACM Symposium on the Theory of Computing 1989)
- Adaboost.M1 (Adaptive boosting) due to Freund and Schapire in 1997.

Adaboost

- The idea of boosting first conceived in 1980's.
  - *Is weakly learnability equivalent to strong learnability?* by Kearns and Valiant (ACM Symposium on the Theory of Computing 1989)
- Adaboost.M1 (Adaptive boosting) due to Freund and Schapire in 1997.
- Adaboost developed for the two-class classification problem, where response coded as -1/1.

Adaboost algorithm for two-class problems

- One class represented by $+1$; other by -1.

Adaboost algorithm for two-class problems

- One class represented by $+1$; other by -1.
- Let each training sample have the same starting weight. If N samples, observation weights $w_i = 1/N$, where $i = 1, 2, \ldots, N$.

Adaboost algorithm for two-class problems

- One class represented by $+1$; other by -1.
- Let each training sample have the same starting weight. If N samples, observation weights $w_i = 1/N$, where $i = 1, 2, \ldots, N$.
- **For** $m = 1$ **to** M **do**:
  - Fit a classifier $G_m(x)$ to training samples using weights w_i.

Adaboost algorithm for two-class problems

- One class represented by $+1$; other by -1.
- Let each training sample have the same starting weight. If N samples, observation weights $w_i = 1/N$, where $i = 1, 2, \ldots, N$.
- **For** $m = 1$ **to** M **do**:
  - Fit a classifier $G_m(x)$ to training samples using weights w_i.
  - Compute the misclassification error (err_m).

Adaboost algorithm for two-class problems

- One class represented by $+1$; other by -1.
- Let each training sample have the same starting weight. If N samples, observation weights $w_i = 1/N$, where $i = 1, 2, \ldots, N$.
- **For** $m = 1$ **to** M **do**:
  - Fit a classifier $G_m(x)$ to training samples using weights w_i.
  - Compute the misclassification error (err_m).
  - Compute the mth stage value given by $\alpha_m = \log\left(\left(1 - \text{err}_m\right)/\text{err}_m\right)$.

Adaboost algorithm for two-class problems

- One class represented by $+1$; other by -1.
- Let each training sample have the same starting weight. If N samples, observation weights $w_i = 1/N$, where $i = 1, 2, \ldots, N$.
- **For** $m = 1$ **to** M **do**:
  - Fit a classifier $G_m(x)$ to training samples using weights w_i.
  - Compute the misclassification error (err_m).
  - Compute the mth stage value given by $\alpha_m = \log\left(\left(1 - \text{err}_m\right)/\text{err}_m\right)$.
  - Set $w_i \leftarrow w_i \cdot \exp\left[\alpha_m \cdot I(y_i \neq G_m(x_i))\right]$, for $i = 1, 2, \ldots, N$.

Adaboost algorithm for two-class problems

- One class represented by $+1$; other by -1.
- Let each training sample have the same starting weight. If N samples, observation weights $w_i = 1/N$, where $i = 1, 2, \ldots, N$.
- **For** $m = 1$ **to** M **do**:
  - Fit a classifier $G_m(x)$ to training samples using weights w_i.
  - Compute the misclassification error (err_m).
  - Compute the mth stage value given by $\alpha_m = \log\left(\left(1 - \text{err}_m\right)/\text{err}_m\right)$.
  - Set $w_i \leftarrow w_i \cdot \exp\left[\alpha_m \cdot I(y_i \neq G_m(x_i))\right]$, for $i = 1, 2, \ldots, N$.
- Output $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

Questions

- Where does the expression for α_m come from? Motivation?

Questions

- Where does the expression for α_m come from? Motivation?
- Can we extend Adaboost to other problems? How?

Questions

- Where does the expression for α_m come from? Motivation?
- Can we extend Adaboost to other problems? How?
- How are we going to re-envision boosting?

Boosting fits an additive model

- Boosting fits an additive expansion in a set of elementary 'basis' functions.

Boosting fits an additive model

- Boosting fits an additive expansion in a set of elementary 'basis' functions.

- **Additive expansion** A basis function expansion of the form

$$f(x) = \sum_{m=1}^{M} \beta_m \, b(x; \gamma_m),$$

where $b(x;\gamma) \in \mathbb{R}$ are 'simple' functions of x, characterized by set of parameters γ_m.

Boosting fits an additive model

- Boosting fits an additive expansion in a set of elementary 'basis' functions.

- **Additive expansion** A basis function expansion of the form

$$f(x) = \sum_{m=1}^{M} \beta_m \, b(x; \gamma_m),$$

where $b(x;\gamma) \in \mathbb{R}$ are 'simple' functions of x, characterized by set of parameters γ_m.

- Can think of a tree as an additive expansion.

How are additive models fit?

- By minimizing **a** loss function averaged over the training data.

How are additive models fit?

- By minimizing **a** loss function averaged over the training data.
- **Loss function minimization**

$$\min_{\{\beta_m,\gamma_m\}} \sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{M} \beta_m\, b(x;\gamma_m)\right)$$

How are additive models fit?

- By minimizing **a** loss function averaged over the training data.
- **Loss function minimization**

$$\min_{\{\beta_m,\gamma_m\}} \sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{M} \beta_m \, b(x;\gamma_m)\right)$$

- Formidable problem in optimization. Also, what does $L(y, f(x))$ look like?

Forward-stagewise additive modeling

- Initialize $f_0(x) = 0$.

Forward-stagewise additive modeling

- Initialize $f_0(x) = 0$.

Forward-stagewise additive modeling

- Initialize $f_0(x) = 0$.
- **For** $m = 1$ **to** M **do**:
  - Compute

$$(\beta_m, \gamma_m) = \arg\min_{\{\beta,\gamma\}} \sum_{i=1}^{N} L\left(y_i, f_{m-1}(x_i) + \beta\, b(x_i;\gamma)\right)$$

.

Forward-stagewise additive modeling

- Initialize $f_0(x) = 0$.
- **For** $m = 1$ **to** M **do**:
  - Compute

$$(\beta_m, \gamma_m) = \arg \min_{\{\beta,\gamma\}} \sum_{i=1}^{N} L\left(y_i, f_{m-1}(x_i) + \beta\, b(x_i;\gamma)\right)$$

.

  - Set $f_m(x) = f_{m-1}(x) + \beta_m\, b(x;\gamma_m)$.

Kinds of loss functions

- Squared error loss (regression)

$$L(y, f(x)) = \frac{1}{2}\left[y - f(x)\right]^2$$

.

Kinds of loss functions

- Squared error loss (regression)

$$L(y, f(x)) = \frac{1}{2}\left[y - f(x)\right]^2$$

.

- Absolute loss (regression)

$$L(y, f(x)) = |y - f(x)|$$

Kinds of loss functions

- Squared error loss (regression)

$$L(y, f(x)) = \frac{1}{2}\left[y - f(x)\right]^2$$

.

- Absolute loss (regression)

$$L(y, f(x)) = |y - f(x)|$$

- Deviance (classification)

$$L(y, p(x)) = -\sum_{k=1}^{K} I(y = G_k) \log p_k(x)$$

Exponential loss and Adaboost

Consider $L(y, f(x)) = \exp\left(-y\, f(x)\right)$.

- For exponential loss functions, additive modeling fitting involves dealing with

$$(\alpha_m, G_m) = \arg\min_{\{\alpha, G\}} \sum_{i=1}^{N} w_i^m \exp\left(-\alpha\, y_i\, G(x_i)\right)$$

Exponential loss and Adaboost

Consider $L(y, f(x)) = \exp(-y\, f(x))$.

- For exponential loss functions, additive modeling fitting involves dealing with

$$(\alpha_m, G_m) = \arg \min_{\{\alpha, G\}} \sum_{i=1}^{N} w_i^m \exp(-\alpha\, y_i\, G(x_i))$$

- **Key idea** Adaboost.M1 is equivalent to forward stagewise additive modeling using the exponential loss function.

Exponential loss and Adaboost (contd.)

The loop in the Adaboost algorithm can be interpreted as:

- Fitting a classification tree to minimize weighted misclassification error, for any value of α.

Exponential loss and Adaboost (contd.)

The loop in the Adaboost algorithm can be interpreted as:

- Fitting a classification tree to minimize weighted misclassification error, for any value of α.
- Given G, estimating α using some algebra and differentiation, resulting in the same expression as in the Adaboost algorithm.

Why exponential loss?

- For a -1/+1 classification problem, the exponential loss function approximates the binomial loss function (i.e., deviance or cross-entropy).

Why exponential loss?

- For a -1/+1 classification problem, the exponential loss function approximates the binomial loss function (i.e., deviance or cross-entropy).
- The additive expansion produced by Adaboost via the exponential loss function estimates

$$\frac{1}{2}\log\frac{\Pr(Y=1|x)}{\Pr(Y=-1|x)}$$

Why exponential loss?

- For a -1/+1 classification problem, the exponential loss function approximates the binomial loss function (i.e., deviance or cross-entropy).

- The additive expansion produced by Adaboost via the exponential loss function estimates

$$\frac{1}{2}\log\frac{\Pr(Y=1|x)}{\Pr(Y=-1|x)}$$

- Exponential loss is quite sensitive to changes in estimated class probabilities.

Boosting trees

- Formally express a tree as

$$T(x;\Theta) = \sum_{j=1}^{J} \gamma_j \, I(x \in R_j),$$

where $\Theta = \{R_j, \gamma_j\}$.

Boosting trees

- Formally express a tree as

$$T(x;\Theta) = \sum_{j=1}^{J} \gamma_j \, I(x \in R_j),$$

where $\Theta = \{R_j, \gamma_j\}$.

- The boosted tree model is a sum of such trees,

$$f_M(x) = \sum_{m=1}^{M} T(x;\Theta_m),$$

induced in a forward stagewise manner.

Boosting trees (contd.)

- We want to solve

$$\hat{\Theta}_m = \text{arg} \min_{\Theta_m} \sum_{i=1}^{N} L\left(y_i, f_{m-1}(x_i) + T(x_i; \Theta_m)\right)$$

where $\Theta_m = \{R_{jm}, \gamma_{jm}\}$.

Boosting trees (contd.)

- We want to solve

$$\hat{\Theta}_m = \text{arg} \min_{\Theta_m} \sum_{i=1}^{N} L\left(y_i, f_{m-1}(x_i) + T(x_i; \Theta_m)\right)$$

where $\Theta_m = \{R_{jm}, \gamma_{jm}\}$.

- To simplify: we want to minimize $L(f)$ with respect to f, where $f(x)$ is constrained to be a sum of trees, and

$$L(f) = \sum_{i=1}^{N} L(y_i, f(x_i)).$$

Gradients and steepest descent

- How do we numerically optimize

$$\hat{\mathbf{f}} = \arg\min_{\mathbf{f}} L(\mathbf{f}),$$

where $\mathbf{f} = \{f(x_1), f(x_2), \ldots, f(x_N)\}$?

Gradients and steepest descent

- How do we numerically optimize

$$\hat{\mathbf{f}} = \text{arg} \min_{\mathbf{f}} L(\mathbf{f}),$$

where $\mathbf{f} = \{f(x_1), f(x_2), \ldots, f(x_N)\}$?

- **Answer** Use the method of steepest descent – an iterative procedure where the function $\mathbf{f}$ is approximated by moving the direction of the negative gradient $(-\mathbf{g}_m)$, where the components of $\mathbf{g}_m$ are

$$g_{im} = \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x_i)=f_{m-1}(x_i)}$$

Issues with this approach

- Steepest/gradient descent is a greedy strategy.

Issues with this approach

- Steepest/gradient descent is a greedy strategy.
- At each step, maximal reduction in the loss function is sought.

Issues with this approach

- Steepest/gradient descent is a greedy strategy.
- At each step, maximal reduction in the loss function is sought.
- This is done by effectively going down the direction of the negative gradient.

Issues with this approach

- Steepest/gradient descent is a greedy strategy.
- At each step, maximal reduction in the loss function is sought.
- This is done by effectively going down the direction of the negative gradient.
- Danger of overfitting! Want to evaluate loss function everywhere, not just at training values.

Gradient tree boosting algorithm for regression

- Initialize $f_0(x) = \arg\min_{\gamma} \sum_{i=1}^{N} L(y_i, \gamma)$.

Gradient tree boosting algorithm for regression

- Initialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$.
- **For** $m = 1$ **to** M **do**:
  - Compute $r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x_i)=f_{m-1}(x_i)}$

Gradient tree boosting algorithm for regression

- Initialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$.
- **For** $m = 1$ **to** M **do**:
  - Compute $r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x_i)=f_{m-1}(x_i)}$
  - Approximate the targets r_{im} by using a regression tree giving terminal regions R_{jm}.

Gradient tree boosting algorithm for regression

- Initialize $f_0(x) = \text{arg}\,\min_{\gamma} \sum_{i=1}^{N} L(y_i, \gamma)$.
- **For** $m = 1$ **to** M **do**:
  - Compute $r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x_i)=f_{m-1}(x_i)}$
  - Approximate the targets r_{im} by using a regression tree giving terminal regions R_{jm}.
  - Compute $\gamma_{jm} = \text{arg}\,\min_{\gamma} \sum_{x_i \in R_{jm}} L\left(y_i, f_{m-1}(x_i) + \gamma\right)$

Gradient tree boosting algorithm for regression

- Initialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$.
- **For** $m = 1$ **to** M **do**:
  - Compute $r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x_i)=f_{m-1}(x_i)}$
  - Approximate the targets r_{im} by using a regression tree giving terminal regions R_{jm}.
  - Compute $\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L\left(y_i, f_{m-1}(x_i) + \gamma\right)$
  - Update $f_m(x) = f_{m-1}(x) + \sum_j \gamma_{jm} I(x \in R_{jm})$

Gradient tree boosting algorithm for regression

- Initialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$.
- **For** $m = 1$ **to** M **do**:
  - Compute $r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x_i)=f_{m-1}(x_i)}$
  - Approximate the targets r_{im} by using a regression tree giving terminal regions R_{jm}.
  - Compute $\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L\left(y_i, f_{m-1}(x_i) + \gamma\right)$
  - Update $f_m(x) = f_{m-1}(x) + \sum_j \gamma_{jm} I(x \in R_{jm})$
- Output $\hat{f}(x) = f_M(x)$.